# Linked Data Architecture for Plan Execution in Distributed CPS

Andrii Berezovskyi, Jad El-Khoury, Elena Fersman

# Linked Data Architecture for Plan Execution in Distributed CPS

Andrii Berezovskyi Jad El-khoury

Elena Fersman

*KTH Royal Institute of Technology, Mechatronics dept.*

**Abstract**

Future cyber-physical systems (CPS) require their components to perform autonomously. To do that safely and efficiently, CPS components will need access to the global state of the whole CPS. These components will require near real-time updates to a subset of the global state to react to changes in the environment. A particular challenge is to monitor state updates from the distributed CPS components: one needs to ensure that only states consistent with the PDDL plan execution semantics can be observed within the system. In order to guarantee that, a component to monitor plan execution is proposed. Microservices based on Linked Data technologies are used to provide a uniform way to access component states, represented as Resource Description Framework (RDF) resources. To ensure the correct ordering of state updates, we present an extension of the OASIS OSLC TRS protocol. Specifically, we strengthen the ordering guarantees of state change events and introduce inlining of the state with the events to prevent state mismatch at the dereferencing stage.

## 1 Introduction

The next wave of the industrial automation, commonly referred to as Industrie 4.0, requires interconnected knowledge-intensive cyber-physical systems (CPS). Tasks that components of such systems need to perform involve collaborative work with human operators as well as cooperation with other CPS components in a continually changing environment. The knowledge gathered from the CPS components can be seen to be part of an enterprise knowledge graph (KG) [8].

Work done by automous CPS components is known as *deliberate acting,* one of the three key motivations of AI research [20, Ch. 1]. Automated planning is fundamental

to enabling deliberate acting by the CPS components. In this paper, we focus on the components that need to act dynamically depending on the state of the world; we capture this need through plan execution. As the official language of the International Planning Competition [6], Planning Domain Definition Language (PDDL) [19] is one of the most popular AI planning languages. In this paper, we chose PDDL to model the domain and its implementations to create plans to be executed. Other AI planning languages include Action Notation Modeling Language (ANML) [31], New Domain Definition Language (NDDL) [9], and others; it is possible to convert PDDL into some of them [11, 23].

A PDDL plan consists of an ordered set of *actions* that can be executed in a step-by-step fashion. A *precondition* contains a condition that shall hold to permit the *action* to be executed. Once executed, the *effect* (or *postcondition*) describes observable changes to the state of multiple real-world objects. Section 3 gives a formal treatment of *preconditions* and *effects*. An example below shows the definition of an *action* of a `Robot` dropping a given `Object` onto a `ConveyorBelt`:

Listing 1: A sample definition of a `dropConveyorBelt` action

```
(:action dropConveyorBelt
  :parameters
    (?cb - ConveyorBelt ?r - Robot
      ?o - Object ?wp - Waypoint)
  :precondition
    (and (carrying ?r ?o)
      (situatedAt ?cb ?wp)
      (on ?r ?wp))
  :effect
    (and (isOn ?o ?cb)
      (not (carrying ?r ?o))))
```

As CPS components are not only distributed but also use different protocols, data formats, information models, it is often necessary to build an integration architecture on top of them in order to pass information around but also to have a *uniform interface* to invoke actions of a plan. Previously, CPS integration was very domain-specific and was in practice tailored to the equipment interfaces at the shop floor via various Field-bus protocols. Recently, new integration standards (many of them web-based) started to appear with the focus on IoT and Instrustrie 4.0 (aka Industrial IoT), including OPC Unified Architecture (OPC-UA) [25], Open Connectivity Foundation [29]. Some of the integration standards are built on the Linked Data principles [22, Ch. 2], such as W3C Web of Things (WoT) [21] and Open Services for Lifecycle Collaboration [1], to mention a few.

Furthermore, CPS systems consist of more distributed and interconnected components than before. This and the need for integration of heterogeneous components pose a need for "companions" on the application level. Such companions commonly take a form of gateways (provide a single point of access to the device API), adaptors (translate

between protocols and data formats), and digital twins (contain a model and keep track of the component state) that are deployed in the cloud.

As described earlier, plan execution relies on tracking the state of the system consistently in order to check whether the *preconditions* and *effects* of the individual *actions* are satisfied. In a distributed architecture with digital twins, this requires the state of the twins to be kept up-to-date with the state of their physical components and exchanged with other twins. Additionally, the certain semantics of plan execution such as the *atomicity* of the executed actions impose an additional restriction on a distributed system architecture where the plans are executed.

The main aim of this paper is to demonstrate how a reliable plan execution can be guaranteed in such a distributed CPS system by developing an underlying integration architecture to support it. The CPS components are integrated using *digital twins* based on Linked Data principles and web standards. A Plan Execution Service component is introduced to ensure only consistent states can be observed by the rest of the components not directly involved in the execution of a plan. We further present the modifications to the OSLC Tracked Resource Set (TRS) protocol [2] for distributing state updates in a CPS system consistenly.

The rest of the paper is structured as follows: in Section 2 we present the use-case centred around the CPS system for the warehouse intralogistics. Section 3 discusses the semantics of PDDL plans and their execution in a distributed system. Section 4 briefly presents OSLC & TRS and discusses the shortcomings of TRS when it comes to plan execution. The implementation is discussed in Section 5 followed by the related work in Section 6 and the conclusion in Section 7.

## 2 Warehouse Intralogistics Use Case

In this section, we present a use case outlining a distributed warehouse environment involving human-machine interaction and warehouse-scale planning.

In a typical warehouse, the following phases of the warehouse operations can be outlined: pre-receipt, receiving, put-away, storage, picking, replenishment, value-adding services and dispatch [30, ch. 3]. The task of a successful warehouse management and control system (WMCS) is to optimise these operations.

The following components are involved in the use-case: the storage is represented by a number of shelves, and the transportation is performed by a conveyor belt, a robotic arm and two robots. Goods are moved along the conveyor belt and loaded onto the robots via a fixed robotic arm. At any point in time, a human can pick an item from the belt for inspection, while the loading process continues. The robots transport the items across the warehouse and unload them onto the shelves.

The CPS components in the warehouse are accessible via their *digital twins*. Digital twins provide a *uniform* interface for the rest of the components to the digital represen-

tation of the component, integrate them with the other heterogeneous components from various vendors via a common standard, among other services.

In the use case, the functions of WMCS are performed by the Warehouse Controller service, assisted by a number of services. The Warehouse Controller keeps track of the warehouse configuration, instantiates digital twins, and translates high-level objectives from the warehouse manager to the lower level planning requests.

While plans themselves are generated with the help of the Plan Generation Service that provides a unified Linked Data frontend to various PDDL planner implementations, the warehouse controller gathers the subset of the up-to-date KG to define the *problem domain* and the *initial state*.

# 3 Plan Execution Syntax and Semantics

In this paper, we deal with *simple plans* without an extension for handling numeric-valued fluents under PDDL 2.1 semantics [19]. The following definitions will be reused from Section 7 of the PDDL 2.1 specification [19] with little simplifications for the purpose of the paper and to account for the unused extensions:

- *Domain* is a tuple $Dom = (Rs, As)$ of finite sets of *relation symbols* and non-durative *actions*.
- *Problem* is a triple $Prob = (Os, Init, G)$ of *domain objects*, the *initial state* specification, and the *goal state* specification.
- $Atms$ is an infinite set of *atoms* that are expressions formed by applying relation symbols to the objects.
- $Init$ is a set of literals formed from the atoms $t \in Atms$
- $G$ is a proposition consisting of atoms.
- *State* $s \subset \mathbb{P}(Atms)$
- A simple *plan* is an *action sequence* $\{a_i\}_{i=0..n}$

Execution of a single action $a_i$ with the effect $e_i$ (which consists of sets of ground atoms $Add_{a_i}$ and $Del_{a_i}$ asserted as positive and negative literals, respectively) causes the state transition $s_i \xrightarrow{e_i} s_{i+1}$ provided that the precondition $Pre_{a_i}$ is satisfied:

$$s_{i+1} := (s_i \setminus Del_{a_i}) \cup Add_{a_i} \quad \text{iff} \quad s_i \vdash p_k \; \forall p_k \in Pre_{a_i} \tag{1}$$

Let $\mathcal{T}$, ranged over by $\tau_j$, denote a finite set of *digital twins*. Each digital twin $\tau_j$ is characterised as an identifier of a physical component $\phi_j$ that it represents and its

current state $\sigma^j$. We shall use $\Phi(\tau_j) : \mathcal{T} \to \Phi$ and $S(t_j) : \mathcal{T} \to S$ to denote the *physical component* and the *current state* of $\tau_j$ respectively.

At any step of the plan execution, the state of the plan consists of the states of all twins:

$$s_i = \bigcup_{j:\tau_j \in T} \sigma_i^j \tag{2}$$

where $\sigma_i^j$ is the (local) state of the twin $\tau_j$ at the plan execution step $i$.

PDDL semantics require the application of the effects to be atomic, while it might not be possible in the real-world system. Consider the example presented in the Introduction. There, the positive postcondition

$$Add_{a_i} = \{isOn(\$o, \$cb)\} \tag{3}$$

will be reflected in the state $\sigma_{i+1}^{cb}$ of the twin $\tau_{cb}$ of the conveyor belt $\$cb$, while the negative postcondition

$$Del_a = \{carrying(\$r, \$o)\} \tag{4}$$

will be reflected in the state $\sigma_{i+1}^{r}$ of the twin $\tau_r$ of the robot $\$r$ (here $\$x$ is used to denote a ground term from $Init$ the variable $?x$ that is bound to when the plan is produced).

Digital Twins $\mathcal{T}$ natually map to cloud-deployed services, thus forming a distributed system. It is possible to observe the distributed system in an intermediary state:

$$s_i \to \sigma_{i+1}^r \cup \sigma_i^{cb} \to s_{i+1} \tag{5}$$

when the robot twin has registered that its physical counterpart is no longer carrying an object, while the conveyor belt twin has not yet reflected that the object has landed on its belt. We will refer to $\sigma_{i+1}^r \cup \sigma_i^{cb}$ as an *intermediary state* $s_i'$.

To generalise the transition between intermediary states, we need a few extra definitions. Let an atom $m_j^+ \in Add_{a_i}$ and let an atom $m_j^- \in Del_{a_i}$. Let a transition function $t_{m_j^-}$ be such that:

$$s_i \xrightarrow{t_{m_j^-}} s_i' \quad s_i' = s_i \setminus \{m_j^-\} \tag{6}$$

Let a transition function $t_{m_j^+}$ be such that:

$$s_i \xrightarrow{t_{m_j^+}} s_i' \quad s_i' = s_i \cup \{m_j^+\} \tag{7}$$

Then, for arbitrary sets $Del_{a_i}$ and $Add_{a_i}$ that have a cardinality of $k$ and $l$, respectively, the following state transition can be made from the state $s_i$:

$$s_i \xrightarrow{t_{m_1^-}} s_i^1 \xrightarrow{t_{m_2^-}} \ldots \xrightarrow{t_{m_k^-}} s_i^k \xrightarrow{t_{m_1^+}} s_i^{k+1} \xrightarrow{t_{m_2^+}} \ldots \xrightarrow{t_{m_l^+}} s_i^{k+l} \tag{8}$$

**Lemma 1.** *The state of the system $s_i^{k+l}$ obtained by successively removing the assertions in $Del_{a_i}$ and adding the assertions in $Add_{a_i}$ is equal to the state $s_{i+1}$.*

$$s_i^{k+l} = s_{i+1} \tag{9}$$

*Proof.*

$$\begin{aligned} s_i^k &= (s_i \setminus \{m_1^-\}) \setminus \{m_2^-\} \ldots \iff s_i^k = s_i \setminus Del_{a_i} \\ s_i^{k+l} &= (s_i^k \cup \{m_1^+\}) \cup \{m_2^+\} \ldots \iff \\ s_i^{k+l} &= s_i^k \cup Add_{a_i} = (s_i \setminus Del_{a_i}) \cup Add_{a_i}) \end{aligned} \tag{10}$$

$(s_i \setminus Del_{a_i}) \cup Add_{a_i}) = s_{i+1}$ by definition (see Equation 1). Thus, $s_i^{k+l} = s_{i+1}$. □

**Lemma 2.** *No intermediary state $s_i^k, 1 \leqslant k \leqslant (n-1)$ is congruent to the state $s_{i+1}$ if none of the assertions in $Add_{a_i}$ or $Del_{a_i}$ are already satisfied in $s_i$.*

*Proof.* Let's assume there exists a state $s_i^k \cong s_{i+1}, 1 \leqslant k \leqslant (n-1)$. The state $s_{i+1}$ can be represented as:

$$s_{i+1} = s_i \wedge \bigwedge_{t_j \in Add_{a_i}} t_j \wedge \bigwedge_{t_j \in Del_{a_i}} \neg t_j \tag{11}$$

Then $s_{i+1} \vdash t_{k+1}$ by construction. If $s_i^k \vdash t_{k+1}$ then $s_i \vdash t_{k+1}$ must have been true as well, which contradicts the requirement in the lemma. Otherwise

$$s_i^k \nvdash t_{k+1} \Rightarrow s_i^k \ncong s_{i+1} \tag{12}$$

□

In order to preserve the semantics of the PDDL plan execution, we must **ensure that none of the intermediary states is observable outside** the plan execution realm.

# 4 Integration Architecture

## 4.1 Linked Data Microservices and Tracked Resource Sets

The state of each component is exposed through microservices that follow the Representational State Transfer (REST) [18, Ch. 5] architectural style to provide clients with a uniform interface, among other advantages. These RESTful microservices further rely on Linked Data principles to enable digital twins and other components to exchange data and their semantics in a machine-readable form no matter what proprietary protocols they use underneath. The use of the Resource Description Framework (RDF) [15] data model allows expressing the concepts from underlying individual information models in a single global ontology or individual ontologies that are subsequently matched. Linked Data microservices follow the OASIS OSLC Core and OSLC Tracked Resource

Set (TRS) specifications [2, 4] to provide a uniform way to perform standard operations on resources within the global state such as create, read, update, delete (CRUD) operations, service discovery capabilities, and tracking of resource changes. Each such Linked Data microservice is also an OSLC Server.

The OSLC TRS specification defines how an OSLC Server can expose an interface for the other components to keep track of the *Change Events* originating from this OSLC Server. The TRS specification distinguishes between three types of Change Events: *creation, modification,* and *deletion* of resources managed by an OSLC Server. A TRS Server (typically running alongside the OSLC Server) appends these Change Events to the TRS Change Log. The Change Log is populated from a certain moment, denoted by a *cut-off event*. The state of the resources managed by the OSLC Server before that event is reflected in a TRS Base.

According to the TRS specification, Clients receive Change Events by *polling* the Change Log regularly. Compared to the push-based approach, polling has the following downsides:

- Polling incurs overhead (both on a client and a server) when a retrieved Change Log contains no new Change Events.
- New Change Events will be received with the delay caused by the *polling interval*; reducing the interval, in turn, increases the overhead even further.

In this paper, we build upon our previously proposed extension [10] to the TRS specification to allow push-based access to the Change Logs, which allows us to alleviate the aforementioned downsides. The resulting architecture is shown in Fig. 1.

Finally, each component assumes the roles of a TRS Server that publishes resource changes and a TRS Client that tracks such changes in other components. The TRS Client in such scenario selectively subscribes to the TRS Servers that allow tracking a relevant subset of a global state.

## 4.2 Mapping Between PDDL Plans and TRS Change Events

The updates to the state of the Digital Twins and the Plan Execution Service, among others, will be published as TRS Change Logs consisting of TRS Change Events. We begin by mapping the ground atoms $t$ of the postcondition to the TRS Change Events:

$$\text{added}(t_i) = \begin{cases} \{\alpha_i\} & \text{if } t_i \in Add_{a_i} \\ \varnothing & \text{otherwise} \end{cases} \tag{13}$$

$$\text{deleted}(t_i) = \begin{cases} \{\delta_i\} & \text{if } t_i \in Del_{a_i} \\ \varnothing & \text{otherwise} \end{cases} \tag{14}$$

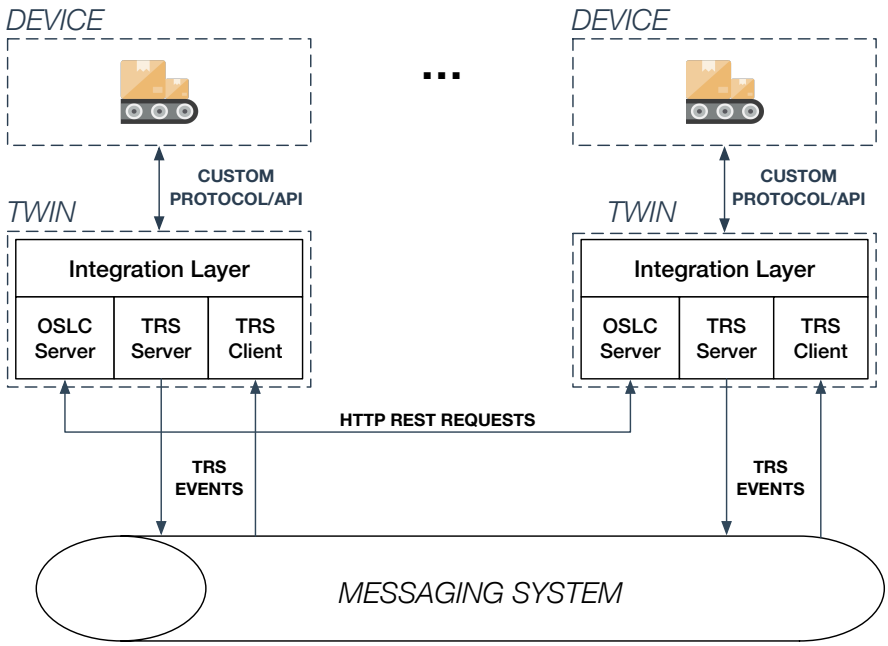


Figure 1: The reference architecture based on Linked Data microservices and publish-subscribe mechanism. TRS Events represent the changes (creation, modification, deletion) to the resources managed by the OSLC Server.

Then, the Change Event can be defined as

$$\mathrm{ce}(t_i) = \mathrm{added}(t_i) \cup \mathrm{deleted}(t_i) \tag{15}$$

Accoring to such definition, function *ce* maps to at most one Change Event:

$$\begin{aligned} &Add_{a_i} \cap Del_{a_i} = \varnothing \\ &\Rightarrow \forall t_i \in Add_{a_i} \cup Del_{a_i} \,:\, |ce(t_i)| = 1 \end{aligned} \tag{16}$$

TRS Change Events $\alpha$ or $\delta$ require an extra step of *dereferencing* a resource.

$$\begin{cases} \mathrm{deref}(\delta_i) := \varnothing \\ \mathrm{deref}(\alpha_i) := \mathcal{KB}_j[\alpha_i] \end{cases} \tag{17}$$

where

- $j = i$ if the tracked resource is embedded in the TRS Change Event, or
- $j \geqslant i$ (this means that for a Change Event with the order $i$ you will either get the resource state at the point of time or any later point of time

During plan execution, Digital Twins may produce Change Events of two kinds:

- PDDL effect-level events
- PDDL action-level events

An effect-level event would be a Change Event for the effect atom `(not (carrying $r1 $b1))` produced by a TRS Server of a Robot Twin. For an effect update event within an action $a_1$, we can write it as $e^k_{i,a1}$. An action-level event for an example presented in the Listing 1 would be "the `pick-up` action completed by $rb_1$". For an action $a_1$, we can write a completion event as $e^{a1}_i$. In order to preserve ordering of action-level events and their atomicity semantics, the two rules shall hold:

1. $t(c^{a_x}_i) > t(c^{a_{x-1}}_i)$ - the action Change Events must be sequentially ordered.
2. $\forall k \ : \ t(c^k_{i,ax}) < t(e^{a_x}_i)$ - all effect Change Events must precede the Change Event for an action completion.

The first rule is obvious; the second rule is needed to prevent a situation when not all effect-level events have propagated and, therefore, the client may observe the system right after an action-level event in a state inconsistent with its effect.

In order to be able to correctly verify whether the precondition of the action $a_i$ is satisfied in order to proceed to execute it, the component must first perform this check at a state right after the execution of the previous action $a_{i-1}$, where the state incorporates the effects from the application of all preceding actions $a_1..a_{i-1}$.

## 4.3 Modifications to the TRS Protocol

Since the state changes need to be communicated via TRS, the TRS protocol needs to be modified for TRS Clients and components of the Digital Twin to make the checks outlined in the previous subsection correctly. To better understand which modifications are needed and why let's consider an example illustrated by Fig. 2.

The figure represents two components $A$ and $B$. Both of them have roles of an OSLC Server, a TRS Server, and a TRS Client, but the figure only illustrates the roles relevant for this example: the OSLC Server and the TRS Server $TS_A$ of the component $A$ and the TRS Client $TC_B$ of the component $B$. When the resource $R_1$ is changed initially in the component $A$ (represented as version $R^{v1}_1$), the corresponding Change Event $CE_1$ is published by its TRS Server $TS_A$. This is followed by the change to the resource $R_2$ and a corresponding Change Event $CE_2$. Subsequently, the TRS Client $TC_B$ receives Change Events $\{CE_1, CE_2\}$ from the TRS Server $TS_A$. But before it manages to dereference the resources $\{R_1, R_2\}$ at the exact versions in which they were at the time of the corresponding

The resources are actually not assumed to be versioned explicitly, version numbers only denote that the resource representation differs in time.

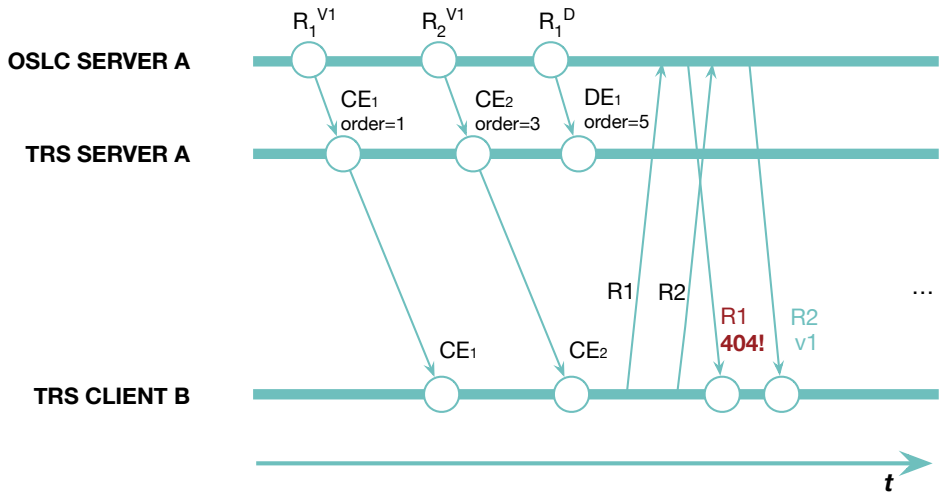


Figure 2: An illustration of a possibly missed state update due to a concurrent resource update.

Change Events, the resource $R1$ happens to be deleted from the server. The TRS Client $TC_B$ receives the resource state $R_1^D$ (as an HTTP *404 Not Found* or *410 Gone* response) when it dereferences it, instead of the expected state $R_1^{v1}$.

Note that even though the Change Event $CE_2$ immediately follows $CE_1$, the TRS protocol only requires their `trs:order` to be increasing. This is not a problem when the Change Events are in a paged HTTP response of a Change Log, because the TRS Client can sort all the changes within a response and assume that it contained all existing Change Events between the smallest and the largest `trs:order`. When we begin distributing Change Events via a messaging system, the TRS Client has no means to establish whether the Change Event with the `trs:order` equal to 2 was lost or never existed[†]. Therefore, our modification to the TRS protocol is to **strengthen the requirement on the `trs:order` property to be *sequential*.** In order to ensure backwards compatibility, these semantics can be attached to a new RDF subproperty `trs:strictOrder` instead. The semantics of `rdfs:subPropertyOf` [5] imply that any resource with a `trs:strictOrder` property also has a `trs:order` with the same value:

> "If a property P is a subproperty of property P', then all pairs of resources which are related by P are also related by P'."

These semantics coupled with the requirements of the OSLC specification not to burden the OSLC clients with the need for any kind of reasoning [7] mean that a compliant OSLC Server will have to ensure that any resource with the `trs:strictOrder` property

[†]A TRS rollback might also cause adjacent Change Events to a have noncontiguous order

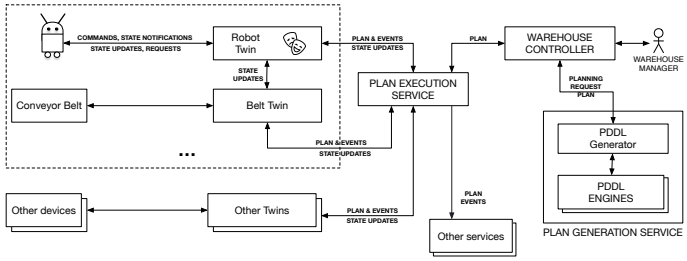


Figure 3: The high-level architecture of the WMCS integration within the use-case. *The dotted line denotes a set of devices and their digital twins involved in the execution of a same plan.*

has the `trs:order` property with the same value *materialised*.

This modification on its own is not enough to ensure every system state observed corresponds to each Change Event because the order of the Change Events is decoupled from the state of the tracked resources that are being *dereferenced*. This is because the Change Events do not contain the tracked resources in question. Using the example above, it would be still possible for the TRS Client $TC_B$ to receive changes $\{CE_1, CE_2\}$ but still to subsequently dereference $\{R_1^D, R_2^{V1}\}$. Therefore, we propose another modification to the TRS protocol to **inline the representation of the tracked resource in the RDF model of a message carrying a `trs:ChangeEvent` resource** at the instant when the Change Event was created in the TRS Server $TS_A$.

With these modifications applied, the TRS Client $TC_B$ will receive tuples of the Change Event resources and the corresponding tracked resources: $\{ (CE_1, R_1^{v1}), (CE_2, R_2^{v1})\}$.

# 5 Implementation

The architecture presented in this paper has been used in the development of the prototype for the use case described in Section 2 as part of the SCOTT project[‡].

The physical components of the use case are represented both by a V-REP simulation scene as well as a pair of Turtlebot robots. The software components were developed using Java, Lua, Python, and Prolog programming languages and were deployed as Docker containers. In particular, the Plan Generation Service was implemented using an OSLC Prolog framework[§], and Digital Twins were developed in Java as JAX-RS services using

[‡]SCOTT (`www.scott-project.eu`) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway.

[§]`https://github.com/EricssonResearch/oslc_prolog`

Eclipse Lyo[¶] framework. The model for the Digital Twins was created using Lyo Designer plugin [17] for Eclipse IDE; Lyo Designer Code Generator [16] was used to generate JAX-RS service skeletons from the model. TRS Change Events were streamed over a Message Queuing Telemetry Transport (MQTT) broker [3]; one topic per TRS Server was used.

The source code is available on Github[‖] under the Apache 2.0 license.

# 6 Related work

Plan execution has been widely studied, but many recent works focused on the architectures with single executor [13, 14, 27]. Munawar et al. [28] proposed an architecture which considers a multi-layer architecture, but the Runtime presented in the paper does not deal with the exchange of the state updates among robots. Kootbally et al. [24] proposed an architecture with direct coupling between the PDDL problem and the state of the system that is kept in a central MySQL database as well as a new Canonical Robot Command Language for executing the PDDL plans and limiting a dependency on a planning language; the possibility of using an out-of-date information is considered. Levine has extensively studied temporal plan execution [26] yet with less focus on the aspects of state exchange between distributed components.

In the area of Linked Data resource updates, various approached exist besides the TRS specification. The most prominent effort is the LDN specification [12], which recently became a W3C recommendation.

# 7 Conclusion

In this paper, we discussed how CPS need to exhibit dynamic behaviour and how can it be represented in a system as plan execution. Next, formalisms for certain key elements of the PDDL language, its plan execution semantics, the TRS specification were presented, and the mapping between PDDL and TRS concepts was performed. An architecture based on Linked Data services was presented for the integration of the CPS components and exchange of state updates among them via the OSLC TRS protocol. In order to guarantee the preservation of the PDDL plan execution semantics, the TRS protocol was extended to support sequential ordering of the Change Events and to inline the Tracked Resources with the Change Events representing changes to them. Furthermore, a Plan Execution Service has been introduced to perform various tasks related to plan execution but also to ensure that the action-level events are not sent until all of the effect-level events have been reported. The system was implemented and made open-

[¶] http://www.eclipse.org/lyo/

[‖] https://github.com/EricssonResearch/scott-eu

source on Github, and the modifications to the TRS protocol will be presented to the relevant OASIS Technical Committee.

There are many directions for future work on the architecture, but we will highlight three main areas: timing (durative actions, sliding windows, time synchronisation between components), concurrency (plans executed in parallel, overlap of the durative actions), and real-world integration (integration with the OPC-UA and other protocols, fault tolerance, replanning).

**Acknowledgements** We thank Leonid Mokrushin for the design and development of the Prolog SDK for OSLC, Yifei Lin for the setup of the ELIoT-based simulation prototype; Aneta Vulgarakis, Konstantinos Vandikas, Rafia Inam for the fruitful discussions regarding the architecture. Icons from Vincent Le Moign, WPZOOM, and ICONS8 were used in the figures.

# References

[1] OSLC Core Version 3.0. Part 1: Overview. https://tools.oasis-open.org/version-control/svn/oslc-core/trunk/specs/oslc-core.html, (Accessed on 2017-10-05)

[2] OSLC Tracked Resource Set specification. https://tools.oasis-open.org/version-control/svn/oslc-core/trunk/specs/trs/tracked-resource-set.html, (Accessed on 2017-09-27)

[3] MQTT Version 3.1.1 (Oct 2014), http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, [Online; accessed 30. Sep. 2018]

[4] OSLC Core version 3.0. OASIS Committee Specification (2017), edited by Jim Amsden. 04 April 2017

[5] RDF Schema 1.1 (Oct 2017), https://www.w3.org/TR/rdf-schema, [Online; accessed 30. Sep. 2018]

[6] International Planning Competition 2018 Classical Tracks, The 28th International Conference on Automated Planning and Scheduling (Oct 2018), https://ipc2018-classical.bitbucket.io, [Online; accessed 7. Dec. 2018]

[7] OSLC Core Version 3.0. Part 6: Resource Shape (May 2018), http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/oslc-core-v3.0-part6-resource-shape.html, [Online; accessed 30. Sep. 2018]

[8] Bakker, R.R.: Knowledge Graphs: representation and structuring of scientific knowledge (1987)

Andrii Berezovskyi and Jad El-khoury are both voting members on the OASIS OSLC Core TC

[9] Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., et al.: EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization. 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) (2012)

[10] Berezovskyi, A., El-khoury, J., Kacimi, O., Loiret, F.: Improving lifecycle query in integrated toolchains using linked data and MQTT-based data warehousing. In: IoT Connected World and Semantic Interoperability Workshop 2017, to appear in LCNS workshop proceedings; arXiv:1803.03525 (2017)

[11] Bernardini, S., Smith, D.E.: Translating PDDL 2.2 into a constraint-based variable / value language (2008)

[12] Capadisli, S., Guy, A., Lange, C., Auer, S., Sambra, A., Berners-Lee, T.: Linked Data Notifications: a resource-centric communication protocol. In: European Semantic Web Conference. pp. 537–553. Springer (2017)

[13] Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtós, N., Carreras, M.: ROSPlan: Planning in the Robot Operating System. In: ICAPS (2015)

[14] Celorrio, S.J., Fernández, F., Borrajo, D.: Integrating planning, execution, and learning to improve plan execution. Computational Intelligence 29, 1–36 (2013)

[15] Cyganiak, R., Wood, D., Lanthaler, M., Klyne, G., Carroll, J.J., McBride, B.: RDF 1.1 concepts and abstract syntax. W3C recommendation 25(02) (2014)

[16] El-khoury, J.: Lyo code generator: A model-based code generator for the development of oslc-compliant tool interfaces. SoftwareX 5, 190 – 194 (2016), http://www.sciencedirect.com/science/article/pii/S2352711016300267

[17] El-Khoury, J., Gurdur, D., Loiret, F., Törngren, M., Zhang, D., Nyberg, M.: Modelling support for a linked data approach to tool interoperability. In: The Second International Conference on Big Data, Small Data, Linked Data and Open Data, ALLDATA, Lisbon. pp. 42–47 (2016)

[18] Fielding, R.T.: Architectural styles and the design of network-based software architectures, vol. 7. University of California, Irvine Doctoral dissertation (2000)

[19] Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. 20, 61–124 (2003)

[20] Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Elsevier (2004)

[21] Guinard, D., Trifa, V.: Building the web of things: with examples in Node.js and Raspberry PI. Manning Publications Co. (2016)

[22] Heath, T., Bizer, C.: Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology 1(1), 1–136 (2011)

[23] Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. 26, 191–246 (2006)

[24] Kootbally, Z., Schlenoff, C., Lawler, C., Kramer, T., Gupta, S.K.: Towards robust assembly with knowledge representation for the planning domain definition language (PDDL) (2014)

[25] Leitner, S.H., Mahnke, W.: OPC UA - Service-oriented Architecture for Industrial Applications. Softwaretechnik-Trends 26 (2006)

[26] Levine, S.J.: Monitoring the execution of temporal plans for robotic systems (2012)

[27] Micalizio, R., Scala, E., Torasso, P.: Intelligent supervision for robust plan execution. In: AI*IA (2011)

[28] Munawar, A., Magistris, G.D., Pham, T.H., Kimura, D., Tatsubori, M., Moriyama, T., Tachibana, R., Booch, G.: Maestrob: A robotics framework for integrated orchestration of low-level control and high-level reasoning. CoRR abs/1806.00802 (2018)

[29] Park, S.: OCF: A New Open IoT Consortium. 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA) pp. 356–359 (2017)

[30] Richards, G.: Kogan Page Publishers (2011), http://app.knovel.com/hotlink/toc/id:kpWMACGIE1/warehouse-management/warehouse-management

[31] Smith, D.E., Frank, J., Cushing, W.: The ANML language (2007)