SOA-PE : A Service-oriented Architecture for Planning and Execution in Cyber-physical Systems

Aneta Vulgarakis Feljan
Ericsson Research, Sweden
aneta.vulgarakis@ericsson.com

Swarup Kumar Mohalik, Mahesh Babu Jayaraman, Ramamurthy Badrinath
Ericsson Research, India
{swarup.kumar.mohalik,mahesh.babu,ramamurthy.badrinath}@ericsson.com

***Abstract*—In this paper, we suggest a service-oriented architecture for planning and execution (SOA-PE) in large scale cyber-physical systems (CPS). SOA-PE provides a clean separation between domain modeling, planning, execution, monitoring and actuation services. This approach helps realize the system-of-systems paradigm allowing the decomposition of system goals into smaller subgoals, thus enhancing the scalability of the proposed solution. In addition to supporting large scale, autonomous systems, the service-oriented approach provides several benefits such as reusability, independent development and deployment, platform independence, transparency and flexibility, to the core services of Planning and Execution in these systems. The architecture targets decentralized, multi-agent systems for solutions like smart transportation and logistics and can scale to larger IoT use cases like smart cities. We illustrate the functionalities of the architecture through a prototype implementation and a case study from the logistics domain.**

I. Introduction

It is widely acknowledged that we are heading to the IoT era, an era where we will see a proliferation of devices that are intended to aid human activity. Correspondingly, their management and operations will increase in complexity, so much so that it will be impossible to manage them with current methodologies. One of the ways to deal with the situation is to create autonomous systems that are self-managed and self-operating. To do this, one has to write intelligent software that leverages the learnings from artificial intelligence (AI) and cognitive technologies research [20]. Further, to industrialize this will require us to create easily composable pieces of software that provide the core building blocks for creating such software systems. In this paper, we describe our work on creating such a piece of software.

Specifically, we define an architecture and implement the components that leverage AI planning and workflow execution technologies. By carefully identifying and modularizing the functionalities necessary for the management and operation of autonomous cyber-physical systems, and exposing these as services, the architecture provides a framework on which domain and problem-specific applications can be developed through composition of the services with minimal glue logic. The approach not only helps in rapid development and deployment, but also scales to large systems, makes maintenance easy and supports enhancements with newer and more efficient algorithms. We call the resulting architecture SOA-PE (Service Oriented Architecture for Planning and Execution).

In the literature, there has been a plethora of works addressing the problem of management and operation of large and complex systems, for instance [2], [17], [22], to name a few. However, these systems are built to target particular problems in specific domains and are not designed to provide generic frameworks for developing applications over them. In [3], a comprehensive utility paradigm for IoT has been proposed that builds upon the IoT-A reference architecture [10] to provide Sensing and Actuation as a Service (SAaaS). One can see the proposed architecture SOA-PE both as an abstraction of SAaaS: that it can be deployed at any level of the system, and also as a refinement: in the sense that the platform provides lower level details of plan synthesis and plan execution services at different level. A service-oriented middleware WebMed has been proposed for cyber-physical systems in [8], but it lacks the planning and plan execution components we believe are necessary for autonomous systems. It may be noted that SOA-PE architecture can be built on top of the communication and data aggregation backbone of WebMed. Other similar efforts include: an architecture geared towards intelligent transportation (T-CPS [26]), extension of SCADA/DCS with service based interactions [11], a service composition framework for CPS [9] and a service-based architecture for CPS with cloud computing with emphasis on security [1]. However, all of these have the same lacuna from planning and plan execution perspective.

One can also view the proposed architecture as a refinement of the MAPE-K loop [7] proposed by IBM. The refinement involves handling the low-level life cycle functions during planning and execution for management. Another architecture which is quite close in spirit to SOA-PE is PELEA – A Domain Independent architecture for Planning, Execution and LEArning [6]. While PELEA is quite comprehensive, we believe that SOA is the right approach for large scale, autonomous systems and provides the typical advantages of SOA, such as reusability, independent development and deployment, platform independence, transparency and flexibility. While in this context, it is necessary to note the cognitive services/API's offered by IBM Watson [24], Microsoft Project Oxford [14] and Google TensorFlow [21]. Currently, these offerings focus on image, speech and natural language recognition tasks, which are not directly related to the planning problem we are addressing, even though one can easily envisage the integration of these capabilities with planning and

execution in real life situations to enrich the final solutions.

In the planning literature, recently there has been a shift in focus from planning to execution [12]. Execution is garnering a central role, thus planning becomes one of the aids for execution. Moreover, it is also realized that instead of generating complex plans, a judicious division of labour between plan and its execution can lead to simpler plans and elegant complete solutions [23]. While developing SOA-PE to provide not just plans, but end-to-end solutions, we became aware of this role of execution acutely. Hence a major feature of SOA-PE is the central position played by the execution runtime that orchestrates majority of the life cycle management APIs.

This paper focuses on describing the overall structure of SOA-PE. Our system supports planning, execution and related services which are intended to work together to address dynamic planning for one or more agents including methods to integrate sensing, monitoring and actuation. The intention is to support a system of systems model where we expect multiple agents to be interacting, each potentially running a similar software stack. The system consumes a modifiable initial state, goals and a behavioral model. The modifiability allows it to consume changes and support the dynamism for operating in the real world. The architecture is pluggable and supports multiple planners and execution engines as the backend services. Our current implementation is an early version and supports PDDL [16] models for planning and simple linear execution. The prototype works on a simple state description of the world embedded inside an application for the use cases we consider. In general this could be a state that is available by query from the world or even as inputs from data processing engines such as an analytics engine.

The paper is organized as follows. We describe the SOA-PE architecture and its components in Section II. In Section III, we outline the critical capabilities of this architecture. The implementation of an early prototype is given in Section IV, with a small case study to illustrate the elements of the architecture. Section V concludes the paper with a summary and pointers to further work.

II. Proposed Architecture

The objective of the proposed architecture SOA-PE is to provide the functionalities related to the broad categories of Sensing, Domain modeling, Planning, Execution, Monitoring, Analytics and Actuation. A typical management and operation life cycle of agents (sensors, actuators, other software/hardware entities) in the system involves each of these functionalities in the following manner. The pre-execution activity involves building the models of domain objects, agents and the action capabilities. When some business objective is to be achieved, one translates them into goals and the domain models are used to synthesize plans to achieve these goals. The plans may be transformed into workflows with domain actions that are then executed by supporting engines. The domain actions are mapped to lower level actuation instructions. The execution is closely monitored to detect any anomalies in the environment assumptions and execution failures. The

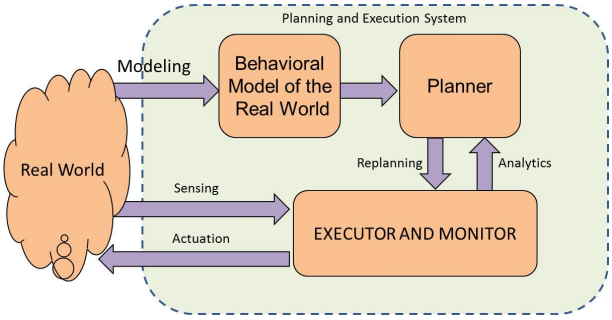

Fig. 1. Lifecycle of Management and Operation in Systems

knowledge of the event and the trend discovered by Analytics is fed back for replanning or change in domain models. Fig 1 gives an overview of the flow in a graphical manner. Note that the work on the critical category of Analytics, which extracts knowledge about the environment and the run-time behaviour of plans, is under progress and hence is not discussed in this paper.

Any architecture that supports the above functionalities must also take into consideration the following characteristics of real-life systems.

- The systems consists of a huge number of agents with differing capabilities.
- Centralized control is often not a choice.
- Agents may be autonomous in nature, which means, they have the capability to (re)plan, execute and monitor their own activities towards their goals.

SOA-PE addresses the above through a service-oriented approach. Roughly, the functionalities are hosted as services and agents can access and compose these services according to their need. Services can also invoke the APIs from other services, providing completely automated flows and extend itself for hierarchical planning/execution using hierarchical domains/problems. The central advantage of the architecture is the separation of concerns, where the agents are responsible for orchestrating the functions according to the needs and the services are responsible for implementing and integrating the internal capabilities, newer algorithms and optimizations. This leads to agile agents and an optimized infrastructure backend providing responsive and efficient management of the system.

The schematic of SOA-PE is given in Fig. 2. In the following, we explain the functions supported by each of the services and their usage.

A. Domain Management Service (DMS)

As the name suggests, the domain management service manages one or more domains registered with this service. It provides APIs to help design the domain models, such as APIs for checking consistency, correctness and completeness of the domain models. It supports what-if analysis and exhaustive verification through functions for simulation and formal-verification respectively. In the case where domain definitions can change, either due to actual change in the agent-action

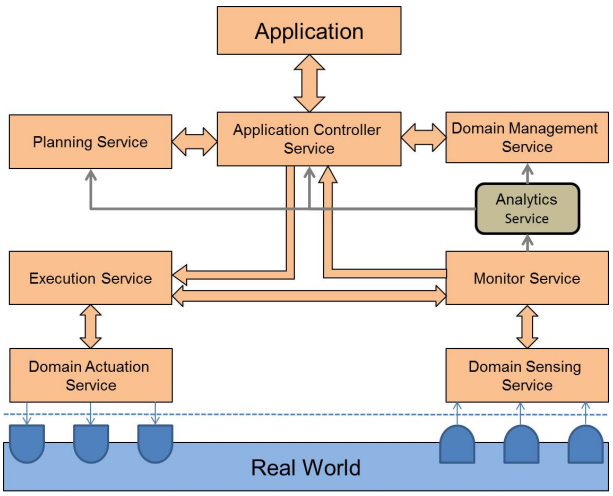

Fig. 2. Overall view of SOA-PE

composition, or due to acquired knowledge about the world, it is necessary to have a verified model before planning and execution. These incremental verification functions are provided by DMS.

One of the main APIs provided by DMS is to prepare filtered domain models to be presented to the planning service for plan synthesis, depending upon the specified goals. This is necessary for scaling the planning algorithms since in most cases the domain definitions cover the agents and actions in the entire domain, whereas the goals involve only a subset of those.

B. Planning Service (PS)

Planning service offers APIs to facilitate the plan synthesis operation. The main function is to take a domain model and a problem description as input and generate a plan as output. Towards this, PS utilizes both domain independent and domain-specific plan synthesis tools. In addition, in order to support the large variety of scenarios in the target system, the APIs for PS include a number of other functions[1]: (1) Selection of planning engine and parameters depending upon the expressibility of the domain specification language and the specific problem, (2) Selection of output format (linear/partial order) depending upon downstream execution capabilities, (3) Translating the output plans to the format expected by an execution service instance, (4) Refactoring of existing plans during replanning, (5) Decomposition and composition of plans for individual agents for distributed/centralized execution resp.

C. Execution Service (ES)

Execution service (ES) offers APIs to facilitate execution of generated plans. In its simplest form, ES has functions to parse the output plan from the planning service and schedule the individual plan steps by invoking the domain actuation service. However, the complexity is largely increased due to many factors such as the following:

- The type of the plans can be different: they can be presented as linear or partial ordered sequence of actions; they can have priorities among the action/agent instances.
- ES has to synchronize with the monitoring/analytics service and stop/resume the current plan execution as and when necessary.
- The agents may be distributed, so ES has to coordinate among the agents. Even in the case where the agents are autonomous, ES has to support the synchronization among agents in case of replanning.
- The action instances in a plan may be one of the following: atomic action that can be directly actuated through the domain actuation service, or goal action to be further decomposed into lower level plans using sub-domains. The ES identifies goal actions and invokes the application controller service with appropriate domains to synthesize the lower level plans.

It is important to note the central role played by the execution service in automatic orchestration of a large part of the operations using other services in SOA-PE. This resonates with the perspective in [23].

D. Domain Sensing Service (DSS)

Domain Sensing Service builds the state of the world as per the domain model. This requires the DSS to interface with underlying layers of abstraction such as IoT Services [5] that in turn, depend upon stacks to process and cleanse raw sensor data, filter the relevant data and extract the semantics using semantic modeling. The prime client for DSS is the Monitor service which regularly calls the DSS to evaluate a set of predicates based on the current world state.

E. Domain Actuation Service (DAS)

Domain actuation service provides the separation between the agents' (devices/systems) actions and the domain action definitions. This allows one to define domains at different abstraction levels depending upon the controllability of the agents. The DAS APIs are invoked to execute a plan step and actuate this using the instructions mapped for the domain actions. Figure 3 shows how the actuation instrumentation is realized through the handshake of ES and DAS. ES checks the pre-conditions for a plan step in the state of the world through the monitor service and calls the DAS API to execute the step. The DAS executes the actuation instructions corresponding to the plan step and returns to ES at completion. At this, ES again checks the effect of the actions on the world through the monitor service to ensure correct execution of the plan step. In case of failures/timeouts, DAS returns to the ES with mutually agreed error codes so that ES can take an appropriate follow-up action. In order to check the completion of step execution, DAS can use either a simple wait-until semantics or more general state-based mechanisms.

[1]Remember that the domain model is prepared by the DMS, the current world state is derived from the Domain Sensing Service and the goal is supplied by an external application (Business Logic or another execution service instance) through the application controller service.

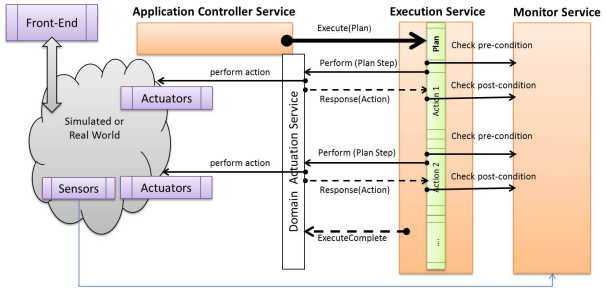

Fig. 3. Actuation Instrumentation Approach

F. Monitor Service (MS)

When a plan is under execution, the monitor service supervises the state of the plan and the state of the world at intervals configured by the application (through the application controller service). This is to ensure that the preconditions of a step holds in the current state before the step is executed through the domain actuation service and to ensure that the executed step caused the expected effect in the state of the world. The overall results - whether success or failure - are conveyed to the execution service to continue/stop execution, to the application controller service to handle in case of failures and to the analytics service to extract any knowledge out of the execution traces.

G. Application Controller Service (ACS)

Application Controller Service (ACS) essentially handles external requests to incorporate new goals in a target system and enforces specified policies for planning, execution and actuation services.

The in-bound requests specify the goals through ACS APIs. The requests may come from business applications or from other execution service instances when executing the goal actions. In a typical response, ACS can invoke the Monitor API to get the current state and then invoke the DMS to get a domain model (possibly filtered). It can then invoke the PS to synthesize a plan and then request the ES to execute the plan.

The ACS APIs allow one to specify the policies for all the services, such as configuring the default planners, timeout values, abort/continue-on-failure, complete/partial replan etc.

III. Capabilities of Proposed Architecture

The cyber-physical systems targeted in the paper are at the extremes of difficulty; they are large scale, heterogenous, distributed, loosely-coupled and are typically system-of-systems. The management and operation of these systems is best achieved through architectures that mirror the above characteristics. SOA-PE leverages the service-oriented nature and exhibits the features that are the right fit for these cyber-physical systems.

In a typical system with networked agents, when a business objective is translated to goals for one or more agents, the agents can use the planning service to synthesize plans. The actions in these plans can be either *atomic actions* or *goal actions* [4]. Atomic actions can be executed using the actuation service directly. However, the processing of goal actions may involve identification of a different domain and a planning service may be invoked to synthesize a plan at a different level. So far as a goal action can be decomposed into a set of simple sub-goals or actuable actions, the hierarchical execution method provided by the execution service leads to a final solution.

In the scenarios where a goal (or, a set of goals) is to be achieved by the actions of a number of agents, the planning engine generates a number of plans, one for each agent, which are then executed by different instances of the execution service. At present, we achieve this by generating a single plan and then decomposing them into local plans. The communication necessary in coordinating the local plans is achieved through communication between the service instances. Note that since the workflow for local plans is different from the global plans, appropriate execution service instances are generated to handle the new workflow.

In real life, the execution of plans do not necessarily lead to the stated goals due to a number of reasons such as (1) violation of the pre-conditions of planned action due to events that were not predicted at the time of plan synthesis, (2) opportunistic changes in the environment that can be exploited to achieve the goals in a better way, (3) changes or enhancements in the goals during the plan execution and (4) the changes in the domain model itself due to acquired knowledge about the domain. The domain actuation service exposes APIs to monitor any of the above changes. In such event, it triggers a number of activities, the principal of them being a replanning call to the planning service. Depending upon the nature of the change, we allow either a complete replanning from the current state to the goal state, or reuse the earlier plan(s) to stitch a plan with minimal change.

A major characteristic of a system-of-systems is the continuous nature of the goals the agents in the systems have to achieve. In order to achieve the continuous goals, the monitoring service checks the completion of a goal and then triggers another execution cycle through the ACS.

We have implemented the architecture using PDDL domains, PDDL planners and an execution service that executes the linear/partial order workflow generated from the plans. However, the framework is generic : so far as a planner supports the domain language and the workflow execution engine supports the plan execution, any combination of domain language, planner and execution engine can be plugged into the system and be used by the applications. For example, an RDDL [18] model can be used to derive a plan using RDDLSIM [19] and the execution service can be used to invoke the recommended action in each step.

IV. Implementation and Case Study

A. A Prototype Implementation

The SOA-PE architecture has been implemented as HTTP services. APIs are realized in the Python language using RESTful approach. PDDL 2.0 has been used as the medium for

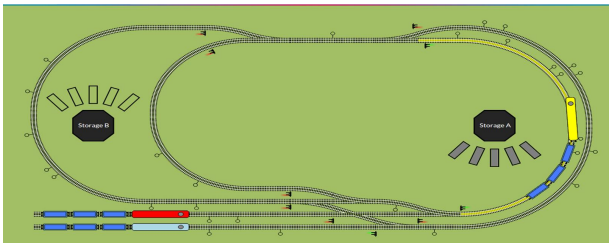

Fig. 4. Railway domain logistics scenario

```
(define (domain TRAIN)
(:requirements :STRIPS :TYPING :FLUENTS
                    :DURATIVE-ACTIONS)
(:types train segment crane package)
(:predicates
            (currentsegment ?t - train ?s - segment)
            (nextsegmentof ?cs - segment ?ns - segment)
            (segmentoccupied ?s - segment)
            (segmentnotoccupied ?s - segment)
            (craneatsegment ?c - crane ?s - segment)
            (packageatsegment ?p - package ?s - segment)
            (packageontrain ?p - package ?t - train) )
(:functions
            (freewagons ?t - train)
            (segmentduration ?s - segment))
(:durative-action move-to-next-segment
:parameters (?t - train ?cs - segment ?ns - segment)
:duration (= ?duration (segmentduration ?cs))
:condition (and
      (at start (currentsegment ?t ?cs))
      .. )
:effect (and
      (at end (currentsegment ?t ?ns) )
      .. ) )

(:action pickup-package
..
(:action drop-package
..
```

Fig. 5. The domain description in PDDL

domain modeling and problem specification. A Command Line Interface (CLI) command layer has been developed to exercise the plan-and-execute functionalities for verification purposes. We have used the planners pddl4j [15] and OPTIC [13] for the planning service. Currently, SOA-PE architecture supports the simplest representation and execution of the plan as a linear sequence of actions. The OPTIC planner output is a timed action sequence which can be simplified to a partial order. The planning service linearizes the generated plan for execution by the ES. In the prototype implementation, application controller server implements domain actuation service endpoints. However, the same can be in an independent server. The domain actuation service currently follows a simple wait-until semantic to monitor the completion of action before it returns to the execution service. Since it is in the experimental state, the DAS has not been interfaced to real world actuation instructions. The execution of plan steps just logs the interaction with the Execution Service and returns.

In the rest of the section, the functioning of the architecture is illustrated using a small case study. The main objective is not to demonstrate the scalability but the functionality of the architecture.

B. Case Study

The case study undertaken is a custom logistics problem in a railway domain. A schematic of the railways tracks, signals and trains is shown in Fig. 4. There are two trains, initially parked at a station. The railway tracks are modeled as a set of contiguous segments. There are fluents modeling the fact whether a segment is occupied by a train or not. The trains can carry different number of packages and these are modeled using numeric fluents. There is a pickup point where there can be different number of packages. There is one crane at the pickup point and another at the drop-off point that have to be scheduled to pickup or drop when the trains are near the cargo points. The problem is defined to pick the packages and drop them at the drop-off point. Since there can be more packages than a train can carry, both the trains are to be utilised to achieve the minimal cost in terms of time for execution. Hence the plans generated for both the trains have to be executed concurrently.

The domain for the scenario is modeled using PDDL. The three actions modeled are *move-to-next-segment, pickup-package* and *drop-package*. The problem is specified using the initial state where there are packages at the pickup point and the goal state where the packages are at the drop point. The planning service is invoked using the problem file and a handle to the domain file. The planner in the planning service outputs a plan to achieve the goal. In this example, we have used the OPTIC planner. The domain, problem and the output plan are shown in Figures 5, 6 and 7 respectively. The generated plan is a sequence of timed steps, each timed step consisting of the start time, the action with the concrete parameters and duration of the step.

The generated plan is executed using the execution service. It breaks down the plan to the individual steps and calls the DAS to implement the low level steps. The DAS is configured to log and print the details of the steps in this illustration.

V. Conclusion and Further Work

The need for an architecture for planning and execution in large, complex, autonomous cyber-physical systems has been addressed in this paper. We have proposed a loosely coupled, flexible service-oriented architecture as a candidate solution. We have outlined how this architecture supports the scale and a number of characteristics of real-life systems such as uncertainty, hierarchical goals and goal revisions and changes in domain models. A prototype of the architecture has been implemented and the end-to-end solution has been illustrated using a realistic case study. With all the components

```
(define (problem TRAIN-01-1)
(:domain TRAIN)
(:objects
        T1 T2 - train
        S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11
        S12 S13 S14 S15 S16 S17 S18 S19 S20 S21 – segment
        P1 P2 P3 – package
        C1 C2 - crane)
(:init
        (currentsegment T1 S1) (currentsegment T2 S2)
        (segmentoccupied S1).... (segmentnotoccupied S21)
        (nextsegmentof S1 S3) ... (nextsegmentof S20 S21)
        (craneatsegment C1 S7) (craneatsegment C2 S12)
        (packageatsegment P1 S7)(packageatsegment P2 S7)
        (= (freewagons T1) 1)(= (freewagons T2) 1) )
        (= (segmentduration S1) 6)..(= (segmentduration S2) 6)
        (= (segmentduration S3) 4)..(= (segmentduration S4) 4)
(:goal      ( and (packageatsegment P1 S12)
              (packageatsegment P2 S12) )) )
```

Fig. 6. Problem specification in PDDL

```
; Plan found with metric 61.013
; States evaluated so far: 924
; States pruned based on pre-heuristic cost lower bound: 0
; Time 1.32
0.000: (move-to-next-segment t2 s2 s6)  [6.000]
0.000: (move-to-next-segment t1 s1 s4)  [6.000]
6.001: (move-to-next-segment t1 s4 s5)  [4.000]
6.001: (move-to-next-segment t2 s6 s8)  [5.000]
10.002: (move-to-next-segment t1 s5 s7)  [5.000]
11.002: (move-to-next-segment t2 s8 s10)  [10.000]
15.003: (move-to-next-segment t1 s7 s9)  [9.000]
15.003: (pickup-package t1 s7 p2 c1)  [0.001]
21.003: (move-to-next-segment t2 s10 s11)  [1.000]
22.004: (move-to-next-segment t2 s11 s12)  [5.000]
27.005: (move-to-next-segment t2 s12 s15)  [6.000]
27.005: (move-to-next-segment t1 s9 s11)  [1.000]
33.006: (move-to-next-segment t2 s15 s19)  [5.000]
33.006: (move-to-next-segment t1 s11 s12)  [5.000]
38.007: (move-to-next-segment t2 s19 s21)  [1.000]
38.007: (move-to-next-segment t1 s12 s15)  [6.000]
39.008: (move-to-next-segment t2 s21 s5)  [2.000]
41.009: (move-to-next-segment t2 s5 s7)  [5.000]
44.006: (drop-package t1 s12 p2 c2)  [0.001]
46.010: (pickup-package t2 s7 p1 c1)  [0.001]
46.010: (move-to-next-segment t2 s7 s9)  [9.000]
55.011: (move-to-next-segment t2 s9 s11)  [1.000]
56.012: (move-to-next-segment t2 s11 s12)  [5.000]
61.013: (drop-package t2 s12 p1 c2)  [0.001]
```

Fig. 7. Generated Plan

of the architecture in place, it is our plan to implement the capabilities hinted in Section III. In particular, these include the synthesis of plans in multi-agent scenarios where the global plan decomposition does not scale, optimal plan merging algorithms to support quick replanning, inclusion of a knowledge acquisition service working in tandem with the plan execution and monitoring to trigger both replanning and possible domain model changes. We also plan to carry out detailed study of performance for large domains to verify our hypothesis on scalability.

REFERENCES

[1] Anees Ara, Mznah Al-Rodhaan,Yuan Tian, Abdullah Al-Dhelaan, *A Secure Service Provisioning Framework for Cyber-Physical Cloud Computing Systems* International Journal of Distributed and Parallel Systems (IJDPS) vol 6(1), January 2015.

[2] Cisco Corp., *The internet of everything for cities*, http://www.cisco.com/web/web/strategy/docs/gov/everything-for-cities.pdf, 2015.

[3] Salvatore Distefano and Giovanni Merlino and Antonio Puliafito, *A Utility Paradigm For Iot: The Sensing Cloud*, Pervasive and mobile computing, vol. 20, 2015.

[4] Ilce Georgievski and Marco Aiello, *An Overview of Hierarchical Task Network Planning*, CoRR, vol. abs/1403.7426, 2014, http://arxiv.org/abs/1403.7426.

[5] Cisco IoT Services, http://www.cisco.com/web/IN/solutions/trends/iot/iot_services.html

[6] Cesar Guzman, Vidal Alcazar, David Prior, Eva Onaindia, Daniel Borrajo, Juan Fdez-Olivares, Ezequiel Quintero, *PELEA: a Domain-Independent Architecture for Planning, Execution and Learning*, In Proceedings of the Scheduling and Planning Applications Workshop, ICAPS, 2012.

[7] IBM Corp., *An Architectural Blueprint for Autonomic Computing*, Technical Report, 2005.

[8] Dat Dac Hoang, Hye-Young Paik, Chae-Kyu Kim, *Service-Oriented Middleware Architectures for Cyber-Physical Systems*, International Journal of Computer Science and Network Security, vol 12(1), January 2012.

[9] Jian Huang, Farokh Bastani, I-Ling Yen, and Wenke Zhang, *A Framework for Efficient Service Composition in Cyber-Physical Systems*, In Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering (SOSE), Nanjing, June 2010.

[10] IOT-A Project Consortium, *Final Architectural Reference Model for the IoT*, Tech. rep., http://www.iot-a.eu/public/public-documents/d1.5/view, 2013.

[11] S. Karnouskos, A.W. Colombo, *Architecting the Next Generation of Service-Based SCADA/DCS : System of Systems*, In Proceedings of IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society, 2011.

[12] Nau, Dana S., Malik Ghallab, and Paolo Traverso. *Blended Planning and Acting: Preliminary Approach, Research Challenges*, In Proceedings the 29th AAAI Conference on Artificial Intelligence, 2015.

[13] OPTIC planner tool, Available online : http://www.inf.kcl.ac.uk/research/groups/PLANNING.

[14] Project Oxford, Microsoft, https://www.projectoxford.ai/

[15] PDDL4J, Available Online, https://github.com/pellierd/pddl4j.

[16] AIPS-98 Planning Competition Committee. *PDDL - The Planning Domain Definition Language. Version 1.2*, In Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS), 1998.

[17] Pecora, Federico, and Amedeo Cesta, *Planning and Scheduling Ingredients for a Multi-Agent System*, In Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands. Vol. 371. 2002.

[18] Scott Sanner, *Relational Dynamic Influence Diagram Language (RDDL): Language Description*, International Probabilistic Planning Conference(IPPC), 2011.

[19] RDDLSIM tool, Available online : http://code.google.com/p/rddlsim/.

[20] Russel, S.J. and P. Norvig, *Artificial intelligence: a modern approach*, Pearson Education, 2003.

[21] Google TensorFlow, http://www.tensorflow.org/

[22] Pablo Valerio, *Amazon Robotics: IoT In The Warehouse*, InformationWeek, http://www.informationweek.com/strategic-cio/amazon-robotics-iot-in-the-warehouse/d/d-id/1322366, 2015.

[23] Aneta Vulgarakis Feljan, Mahesh Babu Jayaraman, Ramamurthy Badrinath, *Towards Dynamic Planning for Cyber Physical Systems*, In Proceedings of the WIP track of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2015.

[24] *IBM Watson*, http://www.ibm.com/smarterplanet/us/en/ibmwatson/.

[25] Daniel S. Weld, Corin R. Anderson, David E. Smith, *Extending Graphplan to Handle Uncertainty & Sensing Actions*, In Proceedings of the 15th National Conference on Artificial Intelligence (AAAI), 1998.

[26] LI Yongfu, SUN Dihua, LIU Weining, ZHANG Xuebo, *A Service-Oriented Architecture for the Transportation Cyber-Physical Systems* In Proceedings of the 31st Chinese Control Conference, July 2012, Hefei, China.