Knowledge-Based Software Management for the Large Scale Rollout of IIoT Applications

Florian Kintzler
Siemens AG
florian.kintzler@siemens.at

Stephan Cejka
Siemens AG
stephan.cejka@siemens.at

***Abstract*—Complex cyber-physical systems (CPS), in which Industrial Internet of Things (IIoT) technology is used, require advanced software maintenance mechanisms to remain dependable and secure. A solution is needed that allows operators to manage software in a CPS not only based on requirements from the information and communications technology (ICT) domain but also from the applications domain. The Knowledge-Based Software Management (KBSM) described in this paper adds an additional layer on top of existing software management systems to integrate the management of the software domain and of the physical domain. This framework was implemented and tested in the smart grid domain as an example for a complex CPS.**

I. Introduction

Cyber-physical systems (CPS) such as the smart grid [1], [2] require secure and resilient mechanisms for the rollout and management of software components for their ever increasing number of managed devices [3]. The smart grid is a critical infrastructure; therefore, security maintenance (e.g., security patches, firewall updates) becomes indispensable. The exchange of operational data uses the same communication channels as for maintenance of the information and communications technology (ICT) domain. Moreover, the smart grid consists of a combination of legacy systems, which are relatively fragile to change, as well as novel devices and software [4]. Such aspects might lead to potential failures in the rollout process, for example, issues related to quality of service in the communication network, faulty hardware, software, or processes [5]. Several deployment processes and application lifecycle management systems for a large scale rollout of software applications in those systems were investigated and evaluated in previous work [6]. However, state-of-the-art deployment frameworks are limited in the types of dependencies they support. This paper introduces a Knowledge-Based Software Management (KBSM) framework built on-top of those frameworks, which is able to include further levels of dependency management for software rollout decisions.

II. Lifecycle Management and Rollout

The investigated use cases from the smart grid domain may require numerous modular applications to be installed on a high number of devices in the field without requiring staff on the field site [7], [8]. They involve several intelligent Secondary Substations (iSSNs) and Building Energy Management Systems (BEMSs) to be managed and supervised by a central control center, for example, by the distribution system operator responsible for the operation of a low voltage grid.

In previous work, several state-of-the-art application lifecycles and deployment processes from the Internet of Things (IoT) domain that seemed suitable for such a system were analyzed, including software distribution tools (Eclipse hawkBit, balena, SWUpdate, Gridlink provisioning framework, iSSN Application Lifecycle Management), application servers (Apache Karaf, Eclipse Virgo), as well as solutions of the leading cloud providers (AWS IoT Greengrass, Azure IoT Edge) [6]. In addition, a generic application lifecycle management framework [9] as well as an OSGi-based deployment framework [6] were implemented. Generally, those systems consist of a central backend and control system in the operator's sphere, responsible for the management of software components on the field devices. They include mechanisms for automated application provisioning, remote and automatic configuration, and update of services. Most of the evaluated frameworks include (at least a subset of) states as depicted in a generic application lifecycle graph in Figure 1 (based on *Arcangeli et al.* [10], the OSGi and the Docker lifecycle graph). The common tasks (e.g., the installation, start, configure, update etc. of software modules; cf. the transitions in Figure 1) could either be initiated manually by a human operator or automatically via (scheduled) scripts. Target devices are connected to and controlled by this backend, with which control messages and software artifacts are exchanged. On the target system side, a device application management executes the received commands (which may also include the download of components) and replies its status to the initiator. By generally utilizing a modular approach (e.g., OSGi modules or Docker/OCI containers), the number of concurrently executed applications even on one device may be high.

III. Level of Dependencies

The analyzed software management systems are designed to roll out software to one or more devices; they need to cover different levels and areas of dependency management. In contrast to consumer IoT use cases, the installation of a software module in a CPS may not affect one device only. Those systems heavily interact with external systems, such as the power grid; thus, unscheduled and unexpected influences are more likely [8]. Therefore, a proper process in a CPS for the rollout of applications to a massive number of devices must take dependencies on various levels into account; beyond

978-1-7281-2989-1/21/$31.00 ©2021 IEEE

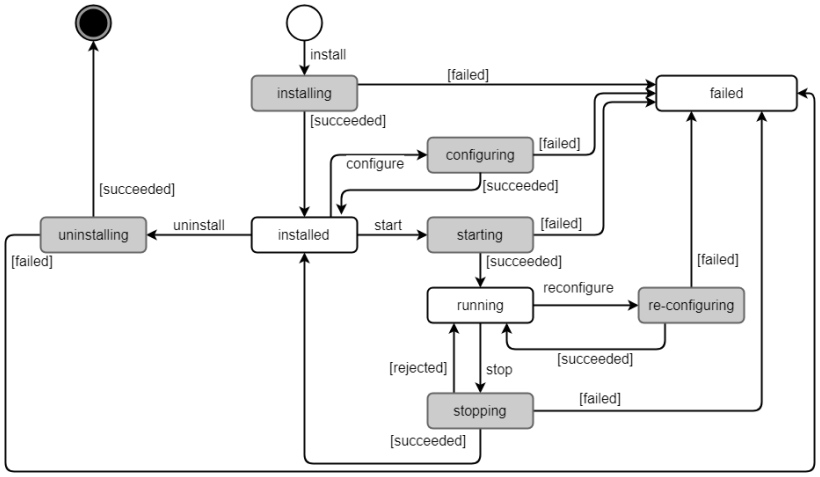

Fig. 1. Generic lifecycle of applications [6]

of what state-of-the-art software rollout systems support. As shown in Figure 2, these interface and functional dependencies range from the device level (dependencies on the applications' runtime environment like drivers, configured sensors etc.) via the system level (protocols, services etc.) up to the domain level (functional dependencies with respect to the controlled physical system; e.g., two controller applications on separate devices should not try to control the same physical parameters). Although dependencies are always resolved to a Yes/No answer, the question itself may contain uncertainties and value ranges (e.g., "Is the voltage level at measurement point X between 220V and 240V?").

To ensure a given level of dependability of the CPS, during and after the software rollout, it is important to cover dependencies on all of these levels, rather than focusing on one of those levels alone. For example, Figure 2 shows that for a rollout of new functionality in a CPS, the management system must on the one hand be able to answer questions on all levels and ensure that the answers to these questions are in a predefined range. On the other hand the system must be able to apply actions on all of these levels.

In general, the dependencies of an application and its impact on the power grid can either be system wide, or limited to a sub-scope of the system. The update of an application that optimizes the energy consumption within a household by using a battery for specific consumers (e.g., garden lightning) but without influence on the power grid, can be rolled out to specific devices without taking other grid components into account. In case the application has an effect on the consumption from the power grid, the update of this application in multiple households connected to the same branch in the low voltage grid might have an effect on the voltage stability of this branch, but is not likely to have an effect beyond the boundaries between low and medium voltage sections of the grid. The application could thus be rolled out in several steps, each step covering a separate low voltage grid section to avoid a blackout of all branches at the same time and to be able to stop and rollback the rollout process in case of failures.

Analogue to transactions in the database domain, the rollout process needs means to automatically rollback the state of the involved distributed devices to a previous state. However, such a complete rollback can only be ensured in the ICT domain. Once an application altered the controlled complex system (e.g., the state of the power grid), the recovery to a

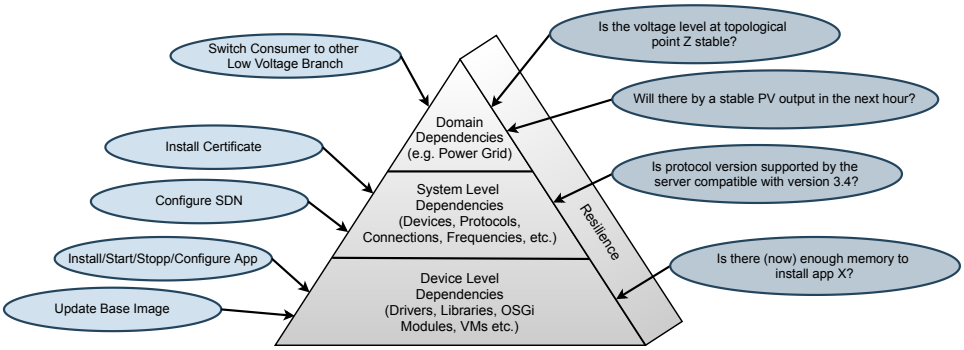

Fig. 2. CPS requirements pyramid. An integrated software management has to be able to answer questions and interact with the system on all levels.

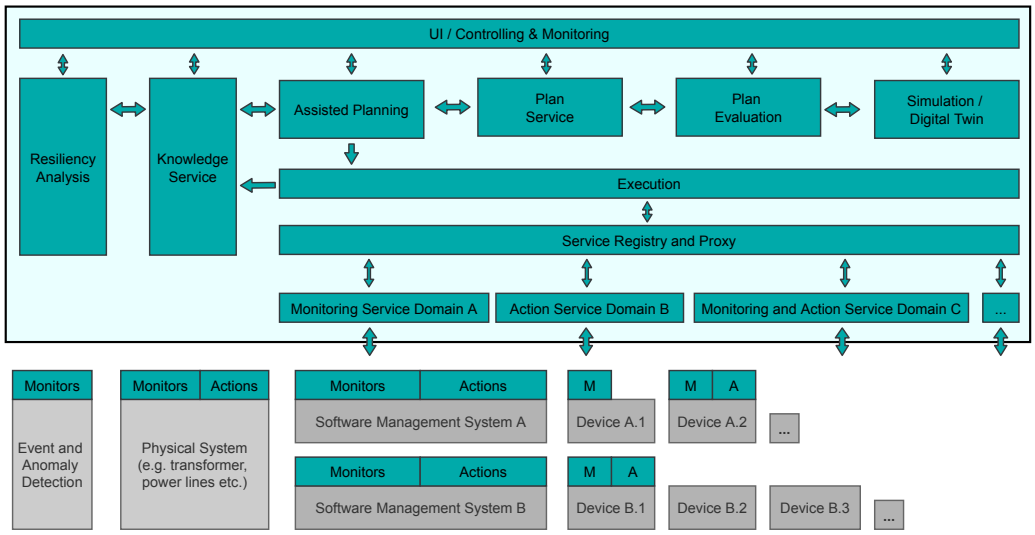

Fig. 3. KBSM Framework. Monitoring and Action Services provide access to the ICP and the physical domain of the CPS.

stable, full-functional state of the overall sub-systems might not be possible. To ensure that an application which is rolled out works correctly, the output of the application could be validated against a model of the environment [8], which may include mathematical models, complex digital twins, and load flow simulations. Once the discrepancy between the real world measurements and its model/twin is above a predefined threshold, a rollback is initiated. This validation cannot be done in fully operational mode only, but also in a warm-up (standby) phase in which the application receives input, but is not yet permitted to provide output to other components and hence, to alter the state of the environment.

If a heterogeneous set of devices is used in a CPS, it is likely to also need a diverse set of software deployment systems, since there are differing requirements for device types with divergent hardware. However, none of the systems analyzed by *Cejka et al.* [6] is able to include knowledge about the physical environment and the functions of the software into a rollout planning and execution process. Examples for these properties include: bandwidth and availability of communication connections, connection of two different controllers to the same physical system (that may lead to possible interferences of control actions), environmental constraints (e.g., weather predictions having an impact on the amount of energy produced by a photovoltaic system), or constraints that could prevent updates of a software component in a given system state. By modeling the different (control) components of the system and monitoring the discrepancies between the model and the real world, control interferences (cf. [11]) and other deviations from the expected behavior can be included into the software update control flow.

IV. Knowledge-Based Software Management

Thus, the various types of dependencies in a CPS as shown in Figure 2 cannot be addressed by use of the previously analyzed evaluated systems alone. To resolve those limitations, an additional layer of system management is proposed which generates and executes software rollout plans on top of existing software management systems and utilizes them to alter the state of the CPS. *Cejka et al.* [6] briefly introduced a Knowledge-Based Software Management system (KBSM), which will be covered in-depth in this contribution. Its components as shown in Figure 3 utilize knowledge about the setup of the CPS and its components for the planning process, including knowledge about the underlying software deployment processes themselves.

A. Monitoring and Action Services

The proposed management component is based on a layer of *Monitoring and Action Services* providing information about the different sub-systems (domains) and the possibility to interact with them.

The *Monitoring Services* can be configured to monitor any state that is necessary to be fulfilled before changes can be made to the systems, including real world sub-systems as well as digital twins and simulation results. Examples in the power system domain include whether the voltage level measured by a given sensor is within a predefined range, whether a given consumer is connected to a predefined branch in the low voltage power grid, or – as a more general example – whether the available memory on a given device exceeds a certain threshold.

The domain-specific *Action Services* provide means to apply changes (i.e., actions) to all parts of the CPS by executing commands. The successful execution is ensured by monitoring the state of the system using the mentioned *Monitoring Services*. Therefore, the definition of the successful installation of an voltage level stabilization algorithm could – besides the successful start of the software itself – also include information of other domains such as the observed voltage level.

The monitoring and actions components can be distributed in the CPS on diverse sets of edge- and IIoT-devices. The *Monitoring and Action Services* are registered at the *Service Registry and Proxy* by descriptions (schemas) of the services they offer, which are used by other components for the creation and execution of plans.

B. Plan Management Service

A system to create and process rollout plans needs a subsystem to store, retrieve and prevent modification of the plans. Thereby, a plan consists of a set of actions to change the state of the CPS to reach a given goal, a set of conditions defining whether and when those actions are executed, and finally a definition of cases in which an execution of the actions is considered to be successful. Figure 4 shows one action that is combined with a set of pre-conditions (defining when an action shall be executed), a set of runtime-conditions (monitoring the parameters of the systems state that should not be violated during the execution, e.g., the maximum time for a software installation or startup) and a set of post-conditions (representing the definition of done). Those conditions can be derived from any of the monitored domains.

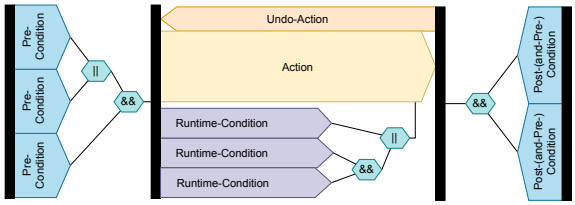

Fig. 4. A plan step consists of logically combined pre-, runtime-, and post-conditions as well as an action and an optional undo-action.

By using a condition as a post-condition for one action and simultaneously as a pre-condition for another one an execution graph emerges that is henceforth called a plan (see subsection IV-G). The graph may have independent sections, but is required to be loop free in order to be able to achieve the desired (sub-)states.

The *Plan Management Service* (KBSM-PS), internally including the plan evaluation and the database, implements an interface to provide functionalities

- to create and update/modify a rollout plan in the database,
- to retrieve a plan from the database,
- to remove a rollout plan from the database,
- to lock/unlock a rollout plan in the database (s.t. it cannot be modified during execution or evaluation), and
- to validate the structure of a plan in the database.

Plans are created either automatically, semi-automatically, or manually; they are stored in the KBSM-PS for being executed or displayed by other modules.

C. Execution

A rollout plan consists of a set of actions that are executed when pre-defined conditions are met. It changes the state of the system by described desired system state transitions which can be monitored. The *Execution* component (KBSM-E) executes a plan by transforming the planned state transitions into concrete actions and by controlling the execution of these actions according to the defined conditions.

Before executing a plan (e.g., on a command originating from the operator via the UI), KBSM-E first retrieves the plan from the KBSM-PS, checks its validity, and transforms it into an execution graph. KBSM-E then starts monitoring the root conditions by initializing and starting monitors for each of the conditions. The monitors return one of four states `UNKNOWN`, `PRE-TRUE` (condition is not yet met, but could be in the future), `TRUE` (the condition is currently met), `POST-TRUE` (the condition is no longer met and will never be met again). Once the logical combination of the pre-monitors of an action becomes `TRUE`, the monitoring of the runtime-conditions and of the post-conditions is started and the action is executed by the according *Action Service*. An action becomes blocked, once the logical combination of its pre-conditions becomes `POST-TRUE`. In that case the operator is notified who decides whether the plan execution shall be stopped or a re-planning (with a potential new goal) shall be started. The current status of all monitors and actions is continuously reported to the operator. KBSM-E uses functionality provided by the external domain-specific *Monitoring and Action Services* as well as the specific underlying rollout processes. It thus provides functionalities

- to start, pause, resume, and stop the execution of a rollout plan,
- to command the execution to ignore blocking issues,
- to report the status of the execution to the operator, and
- to use external, distributed components to monitor and modify the state of the CPS.

D. Knowledge Service

The *Knowledge Service* component (KBSM-KS) contains information about the CPS (including information about the monitoring and action capabilities provided by the *Monitoring and Action Services*) and methods to create a rollout plan. Several different methods were implemented and evaluated (e.g., usage of ontologies to describe the system's components and the relations) to realize a KBSM-KS that uses knowledge about the CPS to derive a rollout plan. Thus, the following properties can be stored in the KBSM-KS: the properties of software components (e.g., size, memory usage), computation devices (e.g., available memory, software environment), the controlled physical system (e.g., for the exemplary use case: tapped transformers, power lines, power switches), the acting roles and the processes executed in the system etc.

Knowledge thus consists of static knowledge (e.g., about software state machines) and dynamic knowledge (e.g., memory usage, running applications). The knowledge graph can be dynamically extended by adding entities and relations from all required domains (e.g., power system, weather, social context). Stored knowledge is domain dependent; new use cases likely require to add new knowledge from other domains.

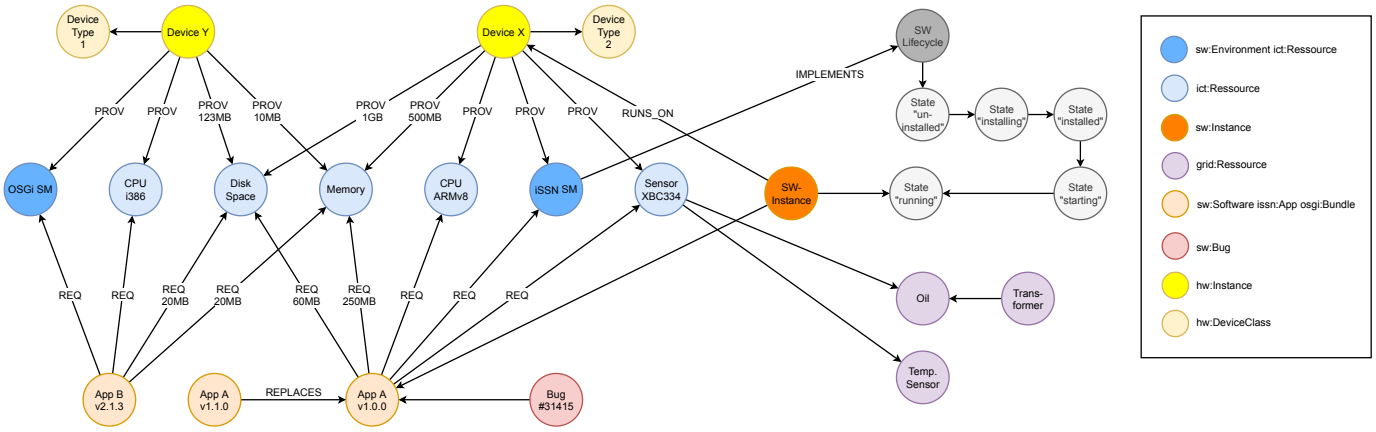

Fig. 5. Subset of a knowledge graph stored in the KBSM-KS

Nevertheless, if use cases share common sub-domains (e.g., weather, ICT) existing knowledge can be reused.

Figure 5 shows a subset of a knowledge graph as stored in the KBSM-KS. In order to derive knowledge from the collected data, a reasoning mechanism was tested. A basic reasoner using the Cypher query language [12] was implemented to evaluate intermediate steps in the process and to integrate complex rules to derive relationships. By utilizing this data preparation, questions occurring during planning can be answered. However, since this cannot be handled independently from the context, the concrete domain has to be taken into account. One of the basic questions to be answered during a rollout planning process is:

> "On which *devices* in my system is the *software* `Y` *installable/startable/ runnable*?"

A generic solution for this question is to make the query dependent on the underlying software management system (SMS) itself. This approach is based on the assumption that there are certain states that all software management lifecycles have in common (cf. Figure 1), including an initial state (S_0) in which the software is not available on the device and a final state in which the software is running in accordance with all requirements (S_n). The path through individual sub-states (S_i with $0 < i < n$) from the initial state S_0 to the final (running) state S_n depends on the specific SMS (e.g., there are systems that distinguish between installation and start, and systems that combine these tasks in one). The question can thus to be transformed to the following set of tasks and questions:

1) Find the definition of the SMS that manages software `Y`.
2) In the definition of the SMS find the initial state S_0.
3) Find the path $S_0 \rightarrow S_1 \rightarrow \ldots \rightarrow S_i \rightarrow S_{i+1} \rightarrow \ldots \rightarrow S_n$ which the software has to pass through until it is in the running state S_n.
4) Find all conditions C_i to pass from state S_i to $S_i + 1$.
5) Derive whether all conditions C_i be fulfilled for a given combination of software `Y` and device D_x.

The set of devices for which the final question can be positively answered is the set to which the software is regarded to be installable.

E. Resiliency Analysis

The *Resiliency Analysis* component (KBSM-RA) uses the stored knowledge about the interactions, roles and processes to derive new knowledge about mis-use cases; i.e., it derives what can go wrong in a system and which actors (human and/or device) might act malicious. Figure 6 shows the steps that such a system has to take to derive new knowledge. As input the process needs a model that contains all system components, their interactions as well as all actors and their interactions with the system's components. Then a control loop graph is derived from this model (depicted in the center of Figure 6) which in turn is the basis for an analysis to derive information on faults, errors and malicious attacks in the system (depicted on the right side of Figure 6). The derived knowledge is stored back into the KBSM-KS to be used for planning and for monitoring the plan execution. This knowledge can then be used in planning to avoid faults and errors, and to prevent situations in which malicious attacks are more likely. The KBSM itself has to adhere to all required security requirements to avoid opening a backdoor into the CPS via the software management. This includes secured communication, authentication and authorization of services and users. In a further step the KBSM-RA could include itself into the analysis process to find vulnerabilities in the overall system including the KBSM.

F. Assisted Planning

The *Assisted Planning* (KBSM-AP) uses stored knowledge to derive a plan of sub-states the system should transition through to create a new system state that is defined by a human operator (e.g., "applications affected by bug #31415 shall be replaced by a newer version", cf. Figure 5). For example, an algorithm generates an optimized plan to roll out new software whereby the impact on the voltage level in the system is kept in predefined limits and the time for the rollout is minimized.

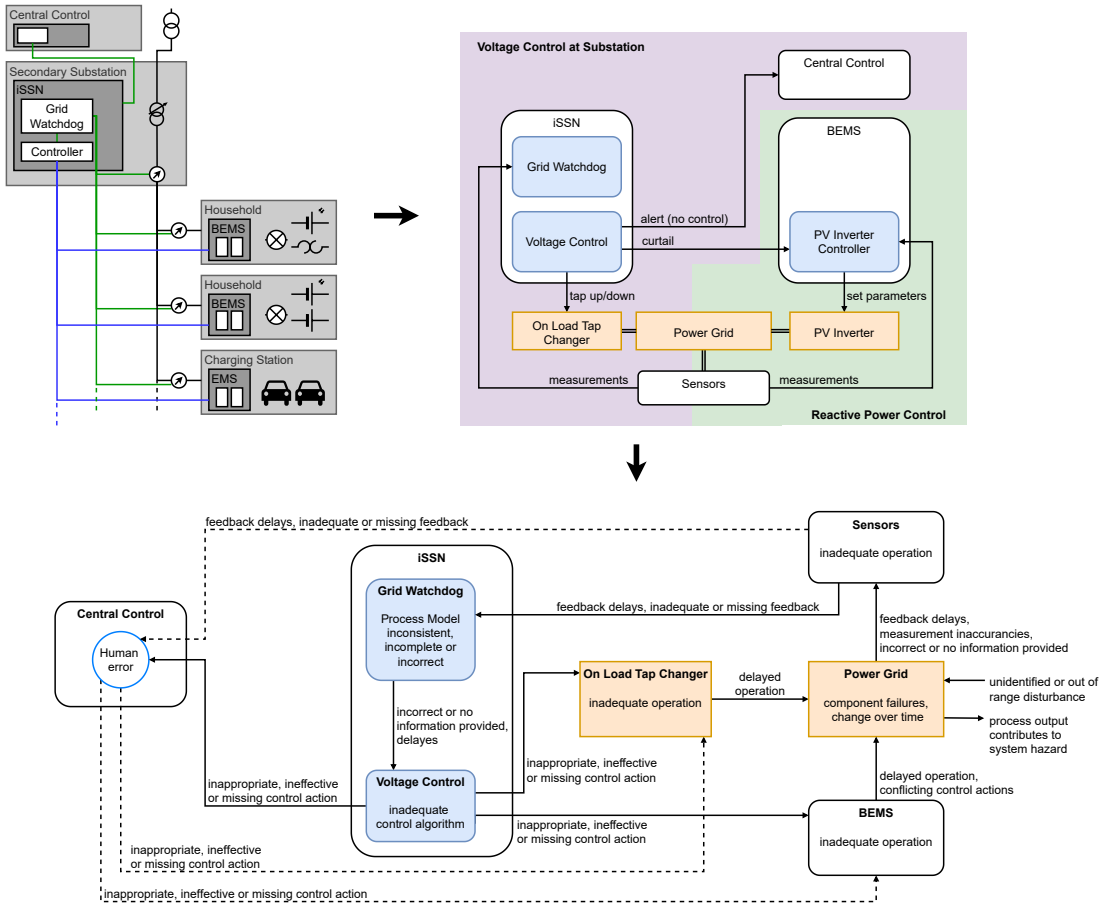

Fig. 6. Example on how a KBSM-RA component can derive new knowledge about the possible errors and failures in the system based on the knowledge about the smart grid topology and an interaction analysis using Systems Theoretic Process Analysis (STPA [13]).

A resilient rollout planning uses knowledge about the topology of the electrical grid to optimize the rollout of a new software version. A generic approach includes properties of the used devices, the ICT infrastructure etc. to provide answers to a wide range of questions for guidance of the operator through the planning process. As a start of such a process the operator needs to know, which parts of the system can (e.g., due to new available software functionality) or should be modified (e.g., due to existing bugs, mis-configurations, or threats). This includes proposals for adapting the structure of the CPS to minimize effects of the rollout which thus needs to include the rollout's timing aspects. The resilient rollout planning answers the question which parts can be updated in parallel and which have to be executed sequentially. The set of further questions this system has to answer include:

- Can an application be installed on a given device?
- Which components should be updated first (e.g., based on threats)?
- What are the time and resource limits from transferring software to a device?
- What are the operational conditions/restrictions under which parts of the functionality can be temporarily deactivated? As examples, updates of photovoltaic inverters should be issued when the sun is currently not shining; e-car charging points should preferably be updated outside of the usual highly frequented time periods.

G. Rollout Planning

The main component of the KBSM is the rollout plan, generated either automatically or semi-automatically by the KBSM-AP (i.e., a user creates the plan with the help of knowledge provided by the KBSM-KS) or manually by an operator using a graphical user interface (Figure 7). The user interface connects to the KBSM-PS to retrieve and display the rollout plans and commits manual changes to the KBSM-PS. Rollout plans can be also generated by external components which add the plans to the KBSM-PS; the UI can also be used to review and manually modify the results of these planning modules (Figure 7). Once a plan is selected in the presented UI, its execution by KBSM-E can be triggered by the user. The KBSM-E then reports back the state of the conditions and actions, which are displayed to the user.

The resilient rollout scheduling algorithm presented by *de Medeiros et al.* [14] (cf. Figure 8) calculates a minimum-time secure rollout plan for software updates for controllable power

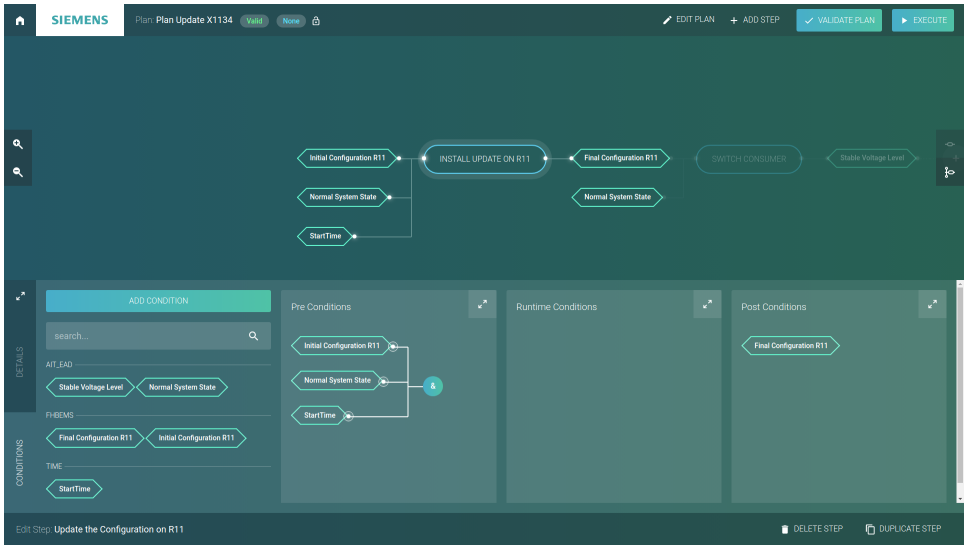

Fig. 7. Plan Editor. Steps can be added, modified or removed by adding actions and conditions and by logically connecting the pre-, runtime-, and post-conditions.

loads. It results in a sequence of device identifier sets. The software on the nodes in one set can be updated in parallel without violating the given conditions (i.e., time and voltage stability). In case the nodes that should be updated are indexed, the result can be represented as an array of index arrays. For example, an array $[[5, 6, 7], [1, 2, 3], [4, 8]]$ would represent a sequence of updates where:

1) Nodes 5, 6 and 7 can be updated together.
2) Nodes 1, 2 and 3 can be updated together, but only when nodes 5, 6 and 7 were updated successfully.
3) Nodes 4 and 8 can be updated together, but only when nodes 1, 2 and 3 were updated successfully.

The sequences that result from the algorithm can be directly mapped to rollout plans that can be executed using KBSM-E.

H. Summary of rollout process steps

In summary, an inter-domain rollout process consists of the following steps:

1) Describe static knowledge about the system, its components, its stakeholders and all known (mis-)use cases.
2) Define standardized sets of failure- and attack-scenarios that can occur during software rollout.
3) Monitor the system to update knowledge about dynamic properties.
4) Add new software and information about the software to the system.
5) Deduce knowledge about security issues, possible failures, and possible mis-use cases.
6) Plan the rollout using the (domain-specific) system knowledge:
   a) Define the goal of the rollout.
   b) Deduce all constraints (from different domains).
   c) Minimize risks and interferences with normal operation (inform the user about non-resolvable risks).
   d) Derive safe-points (i.e., safe and secure intermediate steps).
   e) Include rollback scenarios to safe-points in case a constraint is violated.
7) Evaluate dynamic properties of the planned campaign by using a digital twin (simulation); e.g., execute standardized sets of failure- and attack-scenarios.
8) Rate results and report possible incidents to user.
9) Execute the rollout plan using domain-specific *Monitoring and Action Services*.
10) Perform rollback to a safe-point in case of a constraint violation.

V. Evaluation

To evaluate the effect of a rollout plan resulting from the described planning process, the plan could be executed in a virtual environment in which a simulated CPS replaces the real one. This component mirrors the functional requirements of the KBSM-E component with the exception that the simulation environment has to be controlled by the evaluation component too. The KBSM – built on top of existing SMS – was integrated with the generic application lifecycle management framework [9] as well as the OSGi-based deployment framework [6]. Software rollout commands are issued to those frameworks, that in turn interact with the iSSN and the BEMS field devices. In the investigated use cases the rolled out software is responsible for the power management to stabilize the voltage levels in the power grid controlled by the iSSN as well as the control of parameters of the photovoltaic inverters in the households controlled by the BEMS. The setup was implemented and tested using a Mosaik-based co-simulation [15], [16] as well as in a hardware-in-the-loop lab testbed.

All domain conditions (e.g. voltage levels) and timeouts were correctly detected and the system acted as expected.

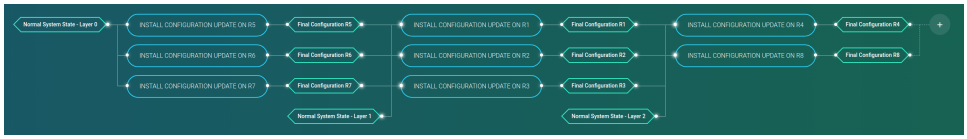

Fig. 8. Generated rollout plan based on the execution sequence derived for the resilient rollout algorithm

Depending on the use-case the implementation of the monitoring and action services turns out to be an extensive effort. However, the system can be dynamically updated during runtime by registering new services, s.t. one can start with the functionality of the underlying SMSs and interactively add new functionality. Even with the support of a graphical interface for the construction of rollout plans, the integration of conditions in these plans for a large CPS is a complex task which should be simplified. One solution could be the usage of domain specific languages (DSL) to describe conditions in a generic way for a given situation to be reused.

VI. Conclusion

The presented KBSM introduces an additional layer on top of existing domain-specific rollout processes as evaluated in previous work [6]. It provides the ability to control those processes by utilizing information about aspects of the CPS which the domain specific software-centered process cannot cover themselves. It further provides means to make the rollout processes dependent on all observable sub-system states of the CPS to be controlled. Covering all those dependencies on all levels from physical to logical level in detail is a challenging task requiring a depth of knowledge about the complex system. In addition to the monitoring capabilities that the KBSM provides, it becomes possible to adapt the state of the CPS so that negative consequences of the software rollout in the CPS can be reduced. Not only the process of rolling out the software was examined, but also the design of the software components to be rolled out.

The KBSM framework was tested with two underlying systems: an OSGi deployment process [6] as well as the iSSN application lifecycle management [9] to rollout software in smart grid use cases. The pursued generic approaches allowed to deliver not only domain specific processes for solving specific problems in the smart grid domain, but also to design and implement a prototype of a system that can be used to resiliently deploy software in any CPS. The knowledge-based approach is currently constrained with respect to the system's dynamic properties. However, in combination with other modeling approaches and the usage of a digital twin to test the software rollout plan against predefined scenarios, it is expected that the presented approach covers a level that increases the dependability of the system during and after the software rollout in comparison to the current state-of-the-art.

References

[1] X. Yu and Y. Xue, "Smart Grids: A Cyber-Physical Systems Perspective," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1058–1070, May 2016.

[2] M. H. Cintuglu, O. A. Mohammed, K. Akkaya, and A. S. Uluagac, "A survey on smart grid cyber-physical system testbeds," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 446–464, 2016.

[3] M. A. Razzaq, S. H. Gill, M. A. Qureshi, and S. Ullah, "Security Issues in the Internet of Things (IoT): A Comprehensive Study," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, 2017.

[4] G. Dileep, "A survey on smart grid technologies and applications," *Renewable Energy*, vol. 146, pp. 2589–2625, 2020.

[5] E. Piatkowska, C. Gavriluta, P. Smith, and F. P. Andrén, "Online Reasoning about the Root Causes of Software Rollout Failures in the Smart Grid," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020, pp. 1–7.

[6] S. Cejka, F. Kintzler, L. Müllner, F. Knorr, M. Mittelsdorf, and J. Schumann, "Application Lifecycle Management for Industrial IoT Devices in Smart Grid Use Cases," in *5th International Conference on Internet of Things, Big Data and Security, IoTBDS 2020, Prague, Czech Republic, 2020*, 2020, pp. 257–266.

[7] M. Faschang, S. Cejka, M. Stefan, A. Frischenschlager, A. Einfalt, K. Diwold, F. Pröstl Andrén, T. Strasser, and F. Kupzog, "Provisioning, deployment, and operation of smart grid applications on substation level," *Computer Science - Research and Development*, vol. 32, no. 1, pp. 117–130, 2017.

[8] F. Kintzler, T. Gawron-Deutsch, S. Cejka, J. Schulte, M. Uslar, E. Veith, E. Piatkowska, P. Smith, F. Kupzog, H. Sandberg, M. Chong, D. Umsonst, and M. Mittelsdorf, "Large Scale Rollout of Smart Grid Services," in *2018 Global Internet of Things Summit*, 2018.

[9] T. Gawron-Deutsch, K. Diwold, S. Cejka, M. Matschnig, and A. Einfalt, "Industrial IoT für Smart Grid-Anwendungen im Feld," *e & i Elektrotechnik und Informationstechnik*, vol. 135, no. 3, pp. 256–263, 6 2018.

[10] J. Arcangeli, R. Boujbel, and S. Leriche, "Automatic deployment of distributed software systems: Definitions and state of the art," *Journal of Systems and Software*, vol. 103, pp. 198–218, 2015.

[11] A. Cervin, "Improved scheduling of control tasks," in *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, 1999, pp. 4–10.

[12] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An Evolving Query Language for Property Graphs," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18, 2018, pp. 1433—-1445.

[13] P. Smith, E. Widl, F. P. Andren, T. Strasser, and E. Piatkowska, "Towards a Systematic Approach for Smart Grid Hazard Analysis and Experiment Specification," in *18th IEEE International Conference on Industrial Informatics (INDIN 2020)*, 2020, pp. 333–339.

[14] M. G. de Medeiros, K. C. Sou, and H. Sandberg, "Minimum-time Secure Rollout of Software Updates for Controllable Power Loads," *Electric Power Systems Research*, vol. 189, p. 106797, 2020.

[15] F. Schloegl, S. Rohjans, S. Lehnhoff, J. Velasquez, C. Steinbrink, and P. Palensky, "Towards a classification scheme for co-simulation approaches in energy systems," in *2015 International Symposium on Smart Electric Distribution Systems and Technologies*, Sep. 2015, pp. 516–521.

[16] C. Steinbrink, M. Blank-Babazadeh, A. El-Ama, S. Holly, B. Lüers, M. Nebel-Wenner, R. P. Ramirez Acosta, T. Raub, J. S. Schwarz, S. Stark, A. Nieße, and S. Lehnhoff, "CPES Testing with mosaik: Co-Simulation Planning, Execution and Analysis," *Applied Sciences*, vol. 9, no. 5, 2019.