Knowledge Driven Approach To Auto-Generate Digital Twins for Industrial Plants

Amar B.[1], Subhrojyoti R. C.[1], Barnali B.[1], Dhakshinamoorthy R.[2], Rajesh N.[2] and Venkatesh C.[3]

[1]TCS Research, 54-B, Tata Research Development and Design Center, Pune, Maharahstra, India, 411013
[2]Tata Consultancy Services Ltd., Siruseri, Chennai, Tamil Nadu 603103
[3]International Institute of Information Technology Hyderabad, Professor CR Rao Rd, Gachibowli, Hyderabad, Telangana 500032

Abstract
Control systems operate industrial plants to accomplish stakeholder objectives like achieving production targets, complying with environmental ordinances, handling faults, etc. Such stakeholder objectives get realised by identifying and executing valid control actions on the plant's control system. E.g., to achieve fault management a *command* is fired to place machines in a *fault mode* when the plant is under an *error* state. Arriving at such control actions is a non-trivial task demanding a detailed understanding of the plant's structure and behaviour. Besides, it is also essential to verify the consequences of such control actions relative to other cross-cutting objectives and plant behaviour. E.g., to fulfil fault management objectives, the action to set machines in *fault mode* may affect production goals due to the machine unavailability. Hence, validation of control actions is vital before executing them using the actual plant's control system. With digital twin technologies (DT), it is now possible to verify the implications of such control actions against a plant's behaviour and objectives in a simulated environment without affecting the actual plant operations. DTs get developed autonomously as one-off solutions to simulate and validate plant control actions in the current state of practice, demanding high efforts and domain expertise. Our paper proposes a knowledge-driven approach enabling automation in DT development. The result of our approach is an auto-generated digital twin that pro-actively mimics the plant's control system behaviour and helps with the validation of control actions before their execution. We use this approach to build three fault management DTs in a power plant. The application of our approach significantly reduces the manual efforts and development time to build such DTs.

Keywords
knowledge-driven engineering, digital twins, control system, knowledge translation, state machines, domain-specific languages

1. Motivation

Modern industrial plants execute multiple processes to accomplish stakeholder objectives like achieving production targets, managing faults, complying with environmental regulations [1], and so on. Process execution involves a control system[2] orchestrating[3] the plant components and sub-systems to produce the desired outcomes for each process. Some of the primary responsibilities of a control system are : 1) Commissioning and integrating the plant components and sub-systems. 2) Identifying and communicating set points to the components. 3) Receiving and processing sensor data. 4) Managing states and modes of components as well as the overall system. 5) Identifying faults and notifying operators.

Control systems enable industrial processes to achieve goals e.g., power generation in thermal power plants, by actuating commands to plant components like pulveriser[4], boiler[5], turbine[5], etc., taking into account their working states and modes. It is a common practice to store the list of such applicable commands in a plant operation manual[6]. The plant operators and engineers refer to such manuals to identify, decide and execute commands through the control system. While executing such commands, the control system ensures that the machines are under appropriate states to accept and process these commands or else flag them back to the operator or engineer as inappropriate actions.

Instructions in operation manuals mostly do not provide any view into the causal effects of command executions on other processes and their objectives, as the number of such scenarios can be too large to incorporate in the operation manual. Plant operators use their judgment to arrive at such actions based on their experience in operating plants over time.

4th WORKSHOP ON KNOWLEDGE-DRIVEN ANALYTICS AND SYSTEMS IMPACTING HUMAN QUALITY OF LIFE (KDAH-CIKM-2021)
amar.banerjee@tcs.com (A. B.); subhrojyoti.c@tcs.com (S. R. C.); barnali.basak@tcs.com (B. B.); dhaks.r@tcs.com (D. R.); rajesh.natesan@tcs.com (R. N.); venkatesh.choppella@iiit.ac (V. C.)

For plants executing multiple processes, manually understanding the implications of each command becomes difficult, insufficient, complex and time-consuming. Emerging digital twin[7] technologies prove useful to reason about such plant behaviour for various what-if actuation scenarios[8].

A digital twin(DT) helps to generate an abstract representation of an operational plant, amenable to human understanding and reasoning. A DT can mimic a plant's behaviour from various aspects, including its control system aspect. This makes it possible for a plant operator to arrive at an appropriate control action, by executing them in a DT and checking their implications against other cross-cutting concerns such as reliability, compliance, etc. DTs execute in a sand-box environment[9] making it possible to try out various actions and observe their implications within the twin without affecting the actual plant. An action found suitable on the twin could then be executed on the existing plant control system. For example, a DT for a power plant can mimic the behaviour of it's individual components like a Boiler, Condenser, Turbine, Steam Outlet etc. and their inter-dependencies E.g. steam is generated in the boiler which is then used to rotate the turbine to produce electricity. Each of the components has it's own specific individual behaviour that contributes to the overall plant behaviour. Such abstract models representing the plant structure, behaviour and processes to enable reasoning is popularly known as a *Digital Twin* of the plant. *Digital* - because it is a virtual model residing in a computer and *Twin* - because it mimics the plants structure and behaviour based on realtime data from the actual plant.

In the current practice, DTs are developed manually for specific aspects of a plant (e.g. asset behaviour monitoring, fault management, energy flow and utilization etc.) in an ad-hoc manner, relying heavily on the knowledge of domain experts. DT development is a collaborative task carried out between domain experts, designers, and developers. The domain experts provide the necessary inputs to construct DT for the aspect of interest and help designers & developers realise it in design and computer programs. The implicit domain knowledge residing with the experts becomes the basis for developing the DT. The implicit nature of knowledge constrains its re-use, making the twin development highly dependent on domain experts and manual efforts. As a result, a DT is typically built from scratch for each problem, adding more time and cost to the plant operations management. A typical plant may need a large number of DTs, can be in hundreds. Hence, a manual approach towards their development can be highly challenging and cost intensive.

This paper proposes a knowledge-driven approach aiming to achieve the following objectives for developing a DT. 1) Enable re-use of domain knowledge in DT development. 2) Provide mechanisms to reduce time and manual efforts in developing DTs.

We implement a framework that exploits the critical idea of domain knowledge explication and it's re-use towards the semi-automated realisation of control system DTs. The main features of the framework are

1) Domain meta-models to explicate plant's domain knowledge in terms of its structure and behaviour
2) Knowledge translation mechanisms to translate plant domain knowledge into a control system model, an abstract representation of the desired DT
3) Completeness and correctness validations on the translated control system model and finally
4) Auto realisation of the final DT from the control system model.

This paper discusses our approach and the results & experiences from applying it to build DTs for three fault management use-cases in a power plant. Our approach shows significant results and the potential to reduce manual efforts by over 50% and development time by almost 60%.

The paper is structured as follows:- in section 2, we discuss related work from the literature on building DTs using knowledge. Section 3 discusses our architecture to generate a DT from knowledge automatically. Section 4 discusses the usage of our approach to semi-automate the development of a DT to support the fault management aspect of an actual plant. The discussion provides initial results from using our approach versus the manual approach previously used by our engineering teams to build similar DTs for an existing plant. Finally, in section 6, we conclude the paper and outline future work.

2. Related Work

This section presents a discussion of current practices, gaps and challenges in DTs development.

The article by Wang et al. studies the application of DTs for fault detection in smart manufacturing[10] units. The main recommendations of their study are:1) Use domain understanding in developing a digital model of the actual plant. 2) Develop data analytic strategies to analyse plant operations based on their objectives, and 3) Create knowledge bases of operation strategies, actions, faults & errors and decisions based on historical data provided by technicians and diagnosticians. . Wang et al.'s approach also proposes primary elements in developing a DT like 1) results from analysis of historical data from plant operations 2) experiences gained from operating plants , 3) knowledge of the experts working in various areas of plant operations, and 4) offline data analysis of operational data to derive new insights. This approach emphasises the importance of gathering knowledge from historical data analysis results, experiences and expert recommendations; however, there is no specific structure suggested in this paper to capture the knowledge. The approach is human-centric and requires manual efforts to develop a DT.

Marmolejo-Saucedo [11] describe a case study on designing and developing a DT for a supply chain process. The approach highlights key enabling technologies to build DTs, such as simulators, constraint solvers, and data analytic tools. Technical and business domains experts primarily configure all of these technologies and tools. Significant manual effort and practical knowledge go into the design of simulators, analytics algorithms, data models, and constraint solvers. This approach focuses on the creation of technology and tool configuration based on experience and knowledge. The implementation of the DT for the supply chain scenario is described as a manual process. The designers use their experience and domain understanding to design the twin. On the other hand, developers create suitable algorithms, data models and configure solvers based on domain understanding. The approach to build the DT stands on manual efforts and implicit knowledge, in their work.

Gabor et al. in [12] propose an architectural approach for cyber-physical systems and safety engineering. The architecture provides a three-model concept to design DTs. As per the authors, physical, cognitive, and contextual(world) models are essential building blocks for a DT. Physical models capture the physical entities and interactions in the cyber-physical system. Cognitive models capture the emergent behaviour of the cyber-physical system from a cognitive aspect. Finally, the contextual model describes the real-world elements affecting the execution of cyber-physical system. The architecture emphasises identifying interactions between these three models to build DTs. The approach uses domain expertise to describe the three models and connect the models based on their interactions in the real world. A significant contribution of this study is to identify the interactions of models that typically work in the back-end of a cyber-physical system. The approach to populate and describe these models remains unaddressed in this study.

A framework based on manufacturing cells (DTMC) by Zhang et al. [13] describes a knowledge and data-driven approach for DT development. The DTMC framework achieves manufacturing automation by enabling intelligence during operations by 1) perceiving data by using analysis approaches, 2) simulating various what-if scenarios to arrive at suitable conclusions, 3) understanding the emergent behaviour of the twin based on data and domain knowledge, 4) predicting future behaviours based on historical data as well as domain knowledge, 5) optimising the executing processes based on domain constraints and the simulation results, and finally 6) implementing strategies to control the plant operations based on reasoning and analysis. The proposed approach relies on identifying the right experts, creating practical learning mechanisms, data analysis, simulation modules, and optimisation approaches to include intelligence in the DT. The critical building block technologies for DTMC implementation are static and dynamic knowledge bases and intellectual skills gathered from experts. The DTMC is a futuristic approach to developing DTs; all the intelligence is still manually gathered and assembled into the twin.

Along with the state of the art approaches, we also study the potential challenges in twin development. Boschert et al. [14] have identified challenges in building future DTs. The authors have also taken an initial step towards defining a next-generation DT. The challenges in building DTs identified in this study are :

1. Integration of multiple simulation technologies - different simulation technologies simulate different aspects of the plants like physics, chemistry, electronics etc. The authors highlight the need to integrate various simulation technologies to develop holistic simulations for the DTs.

2. Changes in context affect plant operation strategies, e.g. the plant area, the network bandwidth, the component units etc. influence the plant operations. Such changes are better understood through a DT that also accounts for context details. Hence, the study emphasises the need to capture and include the context details of the plants that could support extensive reasoning during the plant operations.

3. Addressing real-life problems - The study finally describes the need for DTs to focus on real-life challenges. The primary reason for this is to enhance the exploration of the

problem and solution space. More real-life problems would lead to better domain understanding and gather new knowledge.

The authors suggest addressing the above challenges by working with domain experts and building models, algorithms, and semantic structures for the DTs.

State of the art in developing DTs is highly dependent on manual jobs such as:

1. Gathering domain knowledge from experts.
2. Translating knowledge to simulation environments, models.
3. Developing twin models and software.
4. Relying on domain experts to validate the behaviour of the DT against the actual plant behaviour post-development.

The challenges that arise due to the manual tasks and approaches are: 1) Ensuring the domain understanding of the expert is complete and correct and in sync with the aspect simulated by the DT. E.g. to build a DT for control systems aspect, the knowledge should be relevant and complete with respect to control systems. 2) Ensuring correct translation of domain understanding into computer software. 3) Investing additional efforts for testing and validation of the developed twin. 4) Re-working the whole approach if gaps get identified in knowledge or the knowledge itself is updated, or the implementation technologies get replaced. Overcoming these gaps needs an approach that can assist domain experts to explicate domain knowledge and support developers to directly re-use this knowledge to create DT models and implement the DT.

From our survey of the current state of art and practice of DT development, we see the opportunity to leverage the ideas of a) semantic web [15] and b) model driven engineering (MDE)[16] from the field of computer science, to mitigate the above challenges, as they have mostly been untapped for DT development so far. We see that knowledge about specific domains such as power plants, their structures, behaviour and processes can be captured using semantic web technologies. This enables to deliver much faster, the necessary inputs to the designers and developers of DTs, hence improving knowledge re-use towards domain specific DT development. Model driven engineering, on the other hand, enables synthesis of the knowledge to easily compile and realize them into the DT implementations automatically. This enables harnessing the power of both knowledge-driven as well as model-driven approaches. While semantic knowledge enables querying of relevant domain knowledge at a high level without dependencies on human experts, an implementation domain specific model of a control system, allows reasoning and knowledge-to-code translation for DT realizations using MDE. A key concept exploited in our work is *knowledge translator*. A knowledge translator allows mapping and translating semantic domain knowledge into engineering models of a system, which makes automatic implementation of DTs possible.

The following section discusses our knowledge-driven approach for DT development and addresses the challenges of manual DT development.

3. Knowledge Driven Approach to Auto-generate control system DT

As per the challenges mentioned in the state of practice, we propose a knowledge-driven approach ensuring the reuse of domain knowledge towards the development of DTs in a faster-better-cheaper manner. Our approach results in a framework that allows us to explicate domain knowledge and reuse the same in an automated manner to construct control system DTs for industrial plants. The framework allows capturing knowledge about different aspects of a plant as semantic ontologies. The captured knowledge gets translated into a control system DT through a multi-step translation process. Figure 1 shows a high-level architecture for our proposed framework. We demonstrate our framework on a *fault management* use case in a power plant and discuss the results later in the paper.

The *knowledge-base* is populated by capturing fault-management domain knowledge using a Controlled Natural Language (CNL)[17]. The fault-management knowledge consists of plant components and their fault detection rules, captured in terms of a hierarchy of fault types with detection rules for each type. The captured knowledge then gets translated to a *control system model*[18] that represents the abstract control model of the plant, serving as the key input to build the DT of interest. The translation uses domain-specific translation templates, also stored as part of domain knowledge. The translated *control system model* serves as a formal representation to perform checks to ensure completeness and correctness of the control system model as well as to validate the knowledge from where it is translated. Finally, the *control system model* translates into an implementation using model-to-text[19] translator. In our approach, we use executable finite state machines[20] to implement the final control system

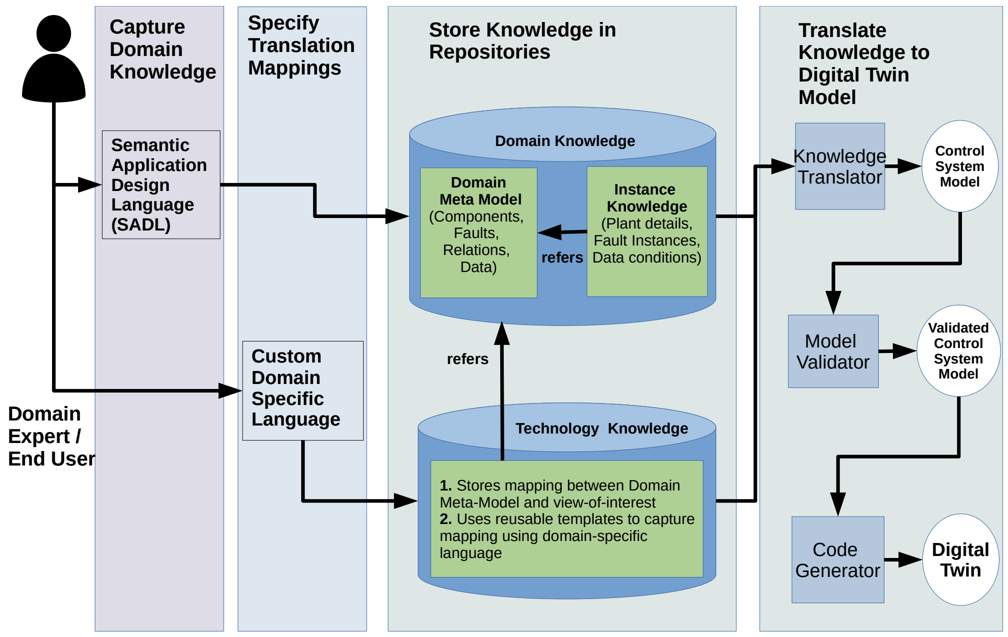

Capture Domain Knowledge
Specify Translation Mappings
Store Knowledge in Repositories
Translate Knowledge to Digital Twin Model
Semantic Application Design Language (SADL)
Domain Knowledge
Domain Meta Model (Components, Faults, Relations, Data)
Instance Knowledge (Plant details, Fault Instances, Data conditions)
refers
Knowledge Translator
Control System Model
Model Validator
Validated Control System Model
refers
Custom Domain Specific Language
Technology Knowledge
1. Stores mapping between Domain Meta-Model and view-of-interest
2. Uses reusable templates to capture mapping using domain-specific language
Domain Expert / End User
Code Generator
Digital Twin

Figure 1: High level solution architecture

DT. In the following sub-sections, we discuss the elements of the solution architecture in detail.

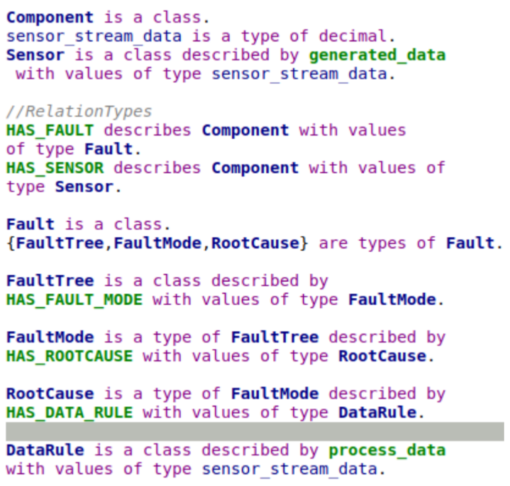
3.1. Describing Knowledge Using Controlled Natural Language

One of the significant challenges in adopting knowledge-driven approaches is providing usable and efficient interfaces to capture knowledge [21]. Web Ontology Language(OWL)[22] is commonly used for describing knowledge in the form of ontologies captured in XML formats. Popular tools like Protege provide GUIs to describe OWL-based ontologies. However, GUI based interfaces make it challenging to capture large ontological structures and reduces human readability significantly[23]. Ideally, knowledge description tools should allow humans to describe and read knowledge with minimal effort using simple and intuitive interfaces. Controlled Natural Languages (CNL) help to mitigate such challenges and can be used for capturing knowledge descriptions[24] using semi-structured English. This makes it easy for experts to describe and read knowledge[25] captured semantically using ontologies.

In our approach, we use Semantic Application Design Language(SADL)[26], a tool that provides English like textual interface for knowledge descriptions, conforming to the idea of a CNL. SADL acts as a front end to populate and store knowledge as OWL-based ontologies. SADL comes with an inbuilt SADL editor that provides support for content assistance, syntax validation, error highlighting, semantic refactoring, and type checking. It uses Apache Jena[27] based reasoners for type checking and validating the described knowledge in the background. It also provides a query language, Simple Protocol and RDF Query Language (SPARQL)[28], to execute knowledge queries[29] and a Semantic Web Rules Language(SWRL)[30] based rule engine to execute semantic rules. SADL syntax[26] allows the description of types, sub-types, properties, relations, constraints, rules as the basic building blocks to describe knowledge. It also provides a feature to import existing OWL models and represent them using English based SADL syntax. SADL reduces the learning curve due to its natural language interface allowing its users to focus on knowledge description. It also makes the knowledge transfer process easier for domain experts.

3.2. Describing Base Ontology Using SADL

The first step to describe knowledge is creating a base ontology. In our approach, we describe the *fault-management* ontology using SADL as shown in figure 2 and 3.



```
Component is a class.
sensor_stream_data is a type of decimal.
Sensor is a class described by generated_data
 with values of type sensor_stream_data.

//RelationTypes
HAS_FAULT describes Component with values
of type Fault.
HAS_SENSOR describes Component with values of
type Sensor.

Fault is a class.
{FaultTree,FaultMode,RootCause} are types of Fault.

FaultTree is a class described by
HAS_FAULT_MODE with values of type FaultMode.

FaultMode is a type of FaultTree described by
HAS_ROOTCAUSE with values of type RootCause.

RootCause is a type of FaultMode described by
HAS_DATA_RULE with values of type DataRule.

DataRule is a class described by process_data
with values of type sensor_stream_data.
```

Figure 2: SADL description of fault knowledge

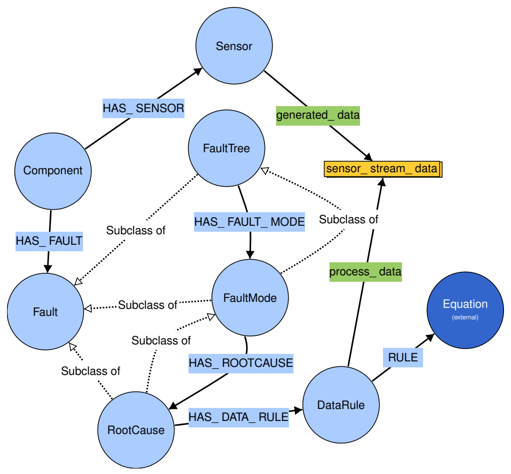


Figure 3: Fault base ontology

A *Component* description in the ontology can have multiple *Faults* represented by the *HAS_FAULT* relation. *RootCause, FaultMode* and *FaultTree* are the subtypes of *Fault*s. A *RootCause* is the lowest level fault that can occur in a *Component*. Multiple *RootCause* faults can be aggregated as a *FaultMode*. Multiple *FaultModes* can be aggregated as *FaultTrees*. *Component*s have *Sensor*s producing *stream_data* which is processed by *DataRule*s. A *DataRule* defines the rules or boolean equations that detect anomalies in *stream_data* to assign a *RootCause*. A *RootCause* fault can be assigned by multiple *DataRules* violations. The base ontology acts like a vocabulary to capture the actual faults and rules in the context of a fault management problem.

3.3. Translation From Knowledge to Control System Model

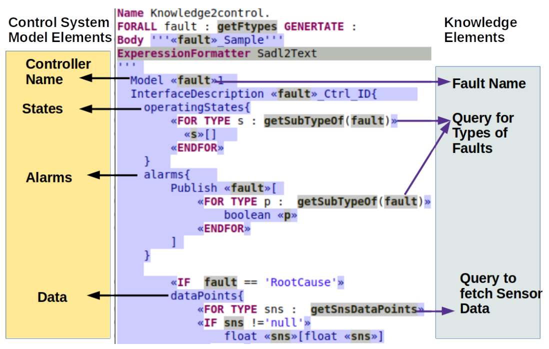
The manual description of domain knowledge may have errors or incompleteness issues. As a result, it becomes necessary to verify if the captured knowledge is good enough to construct a control system DT. To verify the knowledge, we translate it into a control system model and then validate the generated control system model against completeness and consistency checks. To translate domain knowledge into a control system model, we first map concepts from the problem domain into control system concepts using a mapping template implemented using a domain-specific language(DSL). This mapping information is passed onto a translator, which generates the control system fault model from the domain knowledge. We use a discrete control system design language named M&CML[31] to represent the generated control system model. M&CML provides the vocabulary and constructs to design a discrete control system using textual syntax. We discuss the semantics of M&CML separately in the subsequent subsections.

3.3.1. Knowledge Translator

The translator consumes two inputs: 1) knowledge elements from the knowledge base and 2) a translation template. The translation template is created using a DSL that captures the mapping logic from the knowledge elements to the M&CML(control system model) syntax. A knowledge-driven translator uses the mappings to generate a description of the control system model in M&CML. An example of the mapping between knowledge elements and M&CML syntax is shown in figure 4. This approach allows the knowledge-driven translator to be reused for other domains.

A pseudo code describing the internal logic of the translator is shown in algorithm 1.

The generated control system model consists of a hierarchy of controllers where every controller is derived from a fault type in the base ontology. The controllers have operating states that are derived from the instances of the fault types. The hierarchical relation between the controllers is derived from the captured fault propagation knowledge. A structural representation of the generated control system model is shown in figure 5



Control System Model Elements
Controller Name
States
Alarms
Data
Knowledge Elements
Fault Name
Query for Types of Faults
Query to fetch Sensor Data

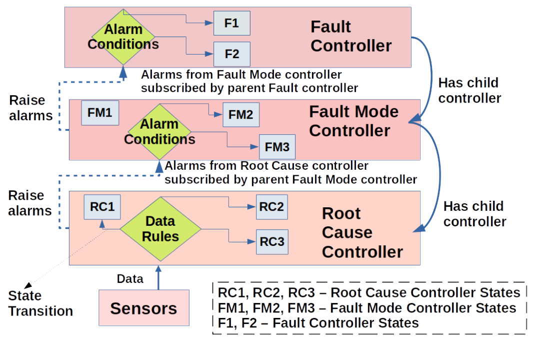
```
Name Knowledge2control.
FORALL fault : getFtypes GENERTATE :
Body '''«fault»_Sample'''
ExperessionFormatter Sadl2Text
'''
  Model «fault»1
  InterfaceDescription «fault»_Ctrl_ID{
    operatingStates{
        «FOR TYPE s : getSubTypeOf(fault)»
          «s»[]
        «ENDFOR»
    }
    alarms{
        Publish «fault»[
            «FOR TYPE p :  getSubTypeOf(fault)»
                boolean «p»
            «ENDFOR»
        ]
    }

        «IF  fault == 'RootCause'»
        dataPoints{
            «FOR TYPE sns :  getSnsDataPoints»
            «IF sns !='null'»
                float «sns»[float «sns»]
```

Figure 4: Translation of fault knowledge to control system model using DSL

Algorithm 1: Translation Algorithm

```
Input: knowledge base, mappings
Output: control model
/* instantiate empty control model */
1 set control model ← empty description
/* iterate through the mappings */
2 for map : mappings do
      /* set ke as knowledge element */
3     set ke ← map.ke
      /* set cs as M&CML syntax from map */
4     set cs ← map.mnc
      /* fetch instances of ke */
5     instances_ke ← fetch instances(ke)
      /* loop over fetched instances */
6     for ike : instances_ke do
          /* append instance to M&CML syntax */
7         set $cs_s yntax$ ← ike.append(cs)
          /* insert cs in control model */
8         set control model.insert($cs_s yntax$)
/* return control model */
9 return control model
```



Alarm Conditions
F1
F2
Fault Controller
Alarms from Fault Mode controller subscribed by parent Fault controller
Has child controller
Raise alarms
FM1
Alarm Conditions
FM2
FM3
Fault Mode Controller
Alarms from Root Cause controller subscribed by parent Fault Mode controller
Has child controller
Raise alarms
RC1
Data Rules
RC2
RC3
Root Cause Controller
State Transition
Data
Sensors
RC1, RC2, RC3 – Root Cause Controller States
FM1, FM2, FM3 – Fault Mode Controller States
F1, F2 – Fault Controller States

Figure 5: Control system model for fault management

3.4. Knowledge and Model Validation

Since the knowledge is described manually by the framework users, there are possibilities for errors or gaps to be present in the knowledge. This poses the risk of such errors propagating into the generated control system model and eventually into the final DT implementation. As the control system model is derived from the knowledge, any gaps in the control system model provide hints about potential gaps in the knowledge. We check the control system model for correctness and completeness, which indirectly validates the knowledge. The validation checks performed on the generated control system model are shown in table 1.

As the control system model gets derived from knowledge and translation templates, any gaps, errors or inconsistencies identified in the model during validation imply gaps in the captured knowledge and translation mappings. In our approach, we rely on the inbuilt validation mechanisms offered by M&CML to perform such validation checks. More checks can easily get added to the control system model by extending the M&CML interpreter. The validation checks ensure that the generated control model is good enough to get used as a DT model and can be used to implement the DT. The validations also ensure the knowledge itself is complete and correct from the control system viewpoint.

3.5. Semantics of Control System Model

We discuss the semantics of the control system model to understand the structure of generated model. The control system model embeds into it semantics of *discrete control systems*. The control system model allows capturing a hierarchical structure of controllers using a parent-child relationship between them. The *InterfaceDescription* block captures the interface(input/output) items for a controller such as *command-response, events or alarms, data, ports, address* and *operating states*. The behaviour for a controller is captured using the *Transition* block. The *Transition* can be specified for various input-output control actions like command-response, events, alarms and data. These semantics provide a basis for the correctness checks of the populated model against the control system viewpoint. M&CML allows designers to describe the control system design using the discussed semantics.

Sr. No	Checks	Description
1	Correctness	It ensures that the generated model follows the semantics of the control system and correctly uses concepts like commands, events, etc. If there are any correctness issues in the control system model like non-terminating cyclic-state-transitions, we can conclude that the knowledge is incorrect from the control system viewpoint. Multiple such checks can be incrementally added in our framework to support the translation process.
2	Completeness	It ensures that the model is complete concerning the control system viewpoint. E.g. A control system design without having a valid state machine description is an incomplete model. Such incompleteness checks can get incorporated into the model checking step based on the needs of the application.
3	Consistency	It ensures that the generated control system model is consistent and does not contain any semantically conflicting terms. For E.g. A state transition rule in the model should not conflict with other transitions. Such consistency checks can also be defined as per the needs of the plant.

Table 1

3.6. Implementing Control System Model as a DT

The generated control system model represents the interactions, behaviour and hierarchical structure of the controllers in the model. However, the generated control system model in M&CML[31] cannot directly execute as a DT. We use SCXML[20], a java-based state machine execution framework, to further translate the control system model into, to realize the final DT. SCXML being a W3C[32] standard, becomes a suitable implementation format for the control system model to translate into. SCXML supports concepts like *states*, *transition*, *input & output events*, *streaming data*, *rule specification* using XML based syntax. As the control system model already captures such information, it becomes possible to easily derive an SCXML based specification from the model. We use model-to-text[19] translation approaches to generate SCXML specifications from the control system model. We use Xtend[33], a Java-based library to implement translation templates for translating the control system model to SCXML.

The correspondence between the control system model and the SCXML state machine structure is shown in figure 6

The *controllers* in the control system model from figure 5 are represented as *hierarchical states* in SCXML. The controller states are represented as *parallel states* inside each of the hierarchical states. The sensors from the control system model are represented as *datasources* in the translated SCXML. The translated SCXML description is an XML file that is executed by the Apache SCXML[34] execution engine. The state machine starts executing from the initial *Start* state that receives and processes data from the sensors. The state machine executes and processes data from the data sources (sensors), and any violation of the rules leads to an appropriate state transition in the state machine. Violation of fault detection rules leads to a transition to the *Root-Cause* state. The state machine moves to a suitable *RootCause* sub-state based on the data source producing erroneous data. The transitions continue till the topmost *Fault* state is reached following the fault aggregation logic.

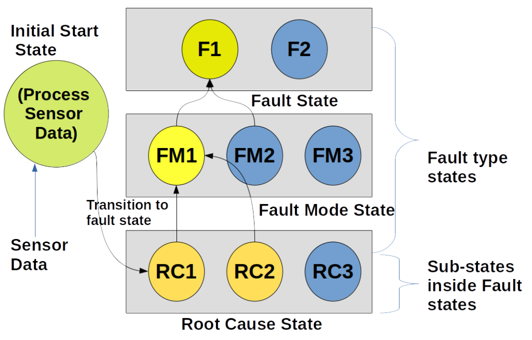


Figure 6: SCXML description of a state-machine structure performing fault management for a control system DT

The SCXML implementation represents the final control system DT. The DT uses the real-time data from the actual data sources and mimics the behaviour of the plant components in terms of their fault hierarchy relations and propagation rules. The generated control system DT can now be used to perform study and analysis of various what-if scenarios like 1) Setting control system in different states. 2) Injecting erroneous data in the state machine and observing the emergent behaviour. 3) Interacting with the state machine by raising injected or dummy events and alarms.

3.6.1. Deploying DTs as a Service

The generated control system DT is deployed as a service in a Java Virtual Environment(JVM). The usage of service-based architecture makes it easier for consumers to interact with the DT. Multiple DTs developed for multiple components are deployed as independent services. The DT services allow consumers to 1) Interact with the DT (e.g. setting states, injecting errors, events and data etc.) using simple service APIs. 2) Observe DT behaviour using API calls. (e.g. getting current state, getting the next state etc.) 3) Integrate DT services with third party analysis platforms. The behaviour analysis and DT usage are consciously kept out of scope in this paper. Our approach provides an automated mechanism to generate the DT implementation from high-level semantic knowledge.

3.7. Summary

Using the knowledge-driven approach, we performed the tasks of 1) capturing knowledge using SADL. 2) translating the knowledge to a control system model represented using M&CML. 3) enabling validations of the control system model for correctness, consistency and completeness. 4) and finally, translating the M&CML based control system model to executable SCXML descriptions. Our approach does not require developers to manually program the DT using general programming languages like C++, Java etc. The translation mechanisms automate the generation of state machine descriptions. SCXML format further reduces the need to write computer programs by providing a configuration based execution engine. The only manual tasks in our approach are describing 1) knowledge using SADL and 2) providing mapping information between domain knowledge and control system concepts as templates created using our DSL. With our approach, it also becomes possible to absorb changes at multiple levels.

1) The knowledge acts as a single point of truth. Any changes in the knowledge can get handled by simply regenerating the DT. Hence there is no manual intervention required to perform refactoring or code changes.

2) Translation templates capture the traceability of knowledge to control system model.

3) Any changes in the control system model can be handled by updating the mapping template, used to translate the control system model into SCXML format and regenerating the DT.

We use this approach for three fault management scenarios to generate DTs for a power plant. The use-case details, results and experiences are discussed in the next section.

4. Use-Case - Power Plant Fault Management

We apply our knowledge-driven framework to generate digital twins for multiple fault managements scenario in a power plant, as discussed below.

1) *Main Steam Temperature Low[35]* - The objective of this scenario is to maintain the main steam temperature at the desired operating range for a given load, coal quality, ambient conditions and detect if the temperature goes below the low threshold and indicate it as a fault.

2) *Main Steam Pressure High[35]* - This scenario maintains the main steam pressure at the desired operating range for a given load, Coal quality and ambient conditions and detects if the temperature goes above the high threshold to indicate a fault.

3) *Mill Outlet Temperature High[36]* - This scenario maintains the main mill outlet temperature within a certain range and detects if the temperature goes beyond the high threshold for a possible fault.

Each of the above scenarios has a fault management structure associated with it. The different fault instances and rules described in SADL are stored as fault knowledge. For other scenarios, the knowledge is described similarly.

Next, we create the mapping template by describing mapping from the captured knowledge to the control system model represented in M&CML. This mapping specification enables the translator to translate the knowledge to generate a control system model in M&CML. Next, we create a model-to-text template using the Xtend library to derive the SCXML specification from the control system model.

4.1. Evaluation & Results

We record the results from applying our approach against the previous efforts of our engineering teams, who manually built similar fault management digital twins for the same power plant. We compare the two approaches using parameters such as development time, number of programmers and lines of code. Table 2 shows the comparison of the manual

SN.	Fault Management Usecase	Hours	Dvlprs	LoC
Manual Effort for Digital Twin Construction				
1	Main Steam Temperature Low	18	2	500
2	Main Steam Pressure High	12	2	300
3	Mill Outlet Temperature High	6	2	300
Knowledge Driven DT Generation				
1	Main Steam Temperature Low	3	1	220
2	Main Steam Pressure High	2	1	140
3	Mill Outlet Temperature High	1	1	90

Table 2
Manual vs Knowledge-driven development of DTs for Power Plant Fault-Management Usecases

approach against our approach. From the comparison, it is evident that our approach significantly outperforms the manual approach by reducing time and efforts for digital twin development. The development time was reduced by almost half and, only half the developers were required while using our approach. We acknowledge that the results are early, and a thorough evaluation of the approach with extensive experimentation is planned going forward. Nevertheless, the early results are promising enough to motivate and establish the applicability of knowledge-driven approaches for developing digital twins.

4.2. Learning and Experiences

We gather the experiences and feedback from the development teams for our approach. The experiences and feedback are as described below:

1) Knowledge description using English is much easier and human friendly. While experts used presentations and text documents to describe the domain knowledge, it became effortless to use the SADL based interface.

2) The captured semantic knowledge is the single point of truth captured from the experts and used towards the automatic realization of the DT through multiple layers of validation and translation.

3) The development team could easily consume the domain knowledge with fewer efforts by reading the SADL descriptions. They could read and understand complex elements like rules, equations in plain English better than the same written using programming languages.

4) The auto-generation of the control system model provided ample time for the verification team to validate the correctness of the model. This resulted in lesser time consumed during post-development testing.

5) it was possible to reuse the knowledge captured for one use-case in other subsequent use-cases too. This further reduced the time to capture knowledge, and the experts did not have to start knowledge descriptions from scratch for each use case.

6) Use of off-the-shelf framework like SCXML allowed the teams to execute the generated SCXML file without writing any code. This resulted in investing more time in verifying the model and the final solution.

7) The teams had to be trained to use the knowledge-driven approach and get them familiar with the techniques. This required some learning curve that is not measured in this study.

5. Conclusion & Future Work

In this paper, we discussed the role of a control system in operating industrial plant systems. We discussed the significance and need for digital twins in arriving at crucial decisions to control and operate complex and mission-critical plants. The state of practice in developing digital twins indicates adhoc approaches to use domain knowledge for developing DTs. The manual approach cannot scale well for large industrial plants, as approaches miss finer details because of informal communications of required domain knowledge that serves as the critical input for DT realisation. Hence, to reduce the time and effort in developing digital twins, we proposed an approach that automates domain knowledge reuse to realise DTs. In our approach, we focus on generating a control system digital twin from domain knowledge. Our approach captures domain knowledge using a controlled natural language. The captured knowledge is translated into a control system model using a knowledge-driven translation approach. Finally, the control system model is used to generate an implementation of the digital twin using the SCXML framework.

Our approach significantly reduces the time and efforts to construct digital twins, as is evident from its usage for a real-life use case discussed in the paper. Our approach can be easily scaled for large and complex systems. Going forward, we will apply our approach in generating digital twins for multiple aspects of a power plant, such as emission compliance, component wear and tear etc. We want to extend the idea of knowledge-driven DT development to other plant systems and enhance it to build digital twins to simulate the behaviour of a system based on the underlying physics and chemistry knowledge.

References

[1] J. J. Downs, E. F. Vogel, A plant-wide industrial process control problem, Computers & chemical engineering 17 (1993) 245–255.

[2] F. Akbarian, E. Fitzgerald, M. Kihl, Synchronization in digital twins for industrial control systems, arXiv preprint arXiv:2006.03447 (2020).

[3] M. Loskyll, I. Heck, J. Schlick, M. Schwarz, Context-based orchestration for control of resource-efficient manufacturing processes, Future Internet 4 (2012) 737–761.

[4] H. Guo, Y. Cheng, T. Ren, L. Wang, L. Yuan, H. Jiang, H. Liu, Pulverization characteristics of coal from a strong outburst-prone coal seam and their impact on gas desorption and diffusion properties, Journal of Natural Gas Science and Engineering 33 (2016) 867–878.

[5] C. Maffezzoni, Boiler-turbine dynamics in power-plant control, Control Engineering Practice 5 (1997) 301–312.

[6] O. Why, DUST-COVERED OPERATIONS AND MAINTENANCE MANUALS, Ph.D. thesis, Doctoral dissertation, Worcester Polytechnic Institute, 2017.

[7] M. Batty, Digital twins, 2018.

[8] W. Kuehn, Digital twins for decision making in complex production and logistic enterprises, International Journal of Design & Nature and Ecodynamics 13 (2018) 260–271.

[9] M. Ayani, M. Ganebäck, A. H. Ng, Digital twin: Applying emulation for machine reconditioning, Procedia Cirp 72 (2018) 243–248.

[10] J. Wang, L. Ye, R. X. Gao, C. Li, L. Zhang, Digital twin for rotating machinery fault diagnosis in smart manufacturing, IJPR 57 (2019) 3920–3934.

[11] J. A. Marmolejo-Saucedo, Design and development of digital twins: a case study in supply chains, Mobile Networks and Applications 25 (2020) 2141–2160.

[12] T. Gabor, L. Belzner, M. Kiermeier, M. T. Beck, A. Neitz, A simulation-based architecture for smart cyber-physical systems, in: 2016 IEEE ICAC, IEEE, 2016, pp. 374–379.

[13] C. Zhang, G. Zhou, J. He, Z. Li, W. Cheng, A data-and knowledge-driven framework for digital twin manufacturing cell, Procedia CIRP 83 (2019) 345–350.

[14] S. Boschert, C. Heinrich, R. Rosen, Next generation digital twin, in: Proc. tmce, Las Palmas de Gran Canaria, Spain, 2018, pp. 209–218.

[15] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific american 284 (2001) 34–43.

[16] S. Kent, Model driven engineering, in: International conference on integrated formal methods, Springer, 2002, pp. 286–298.

[17] H. Safwat, M. Zarrouk, B. Davis, Rewriting simplified text into a controlled natural language (2018).

[18] P. Patwari, A. Banerjee, S. R. Chaudhuri, S. Natarajan, Learning's from developing a domain specific engineering environment for control systems, in: Proceedings of the 9th India Software Engineering Conference, 2016, pp. 177–183.

[19] M. Albert, J. Muñoz, V. Pelechano, Ó. Pastor, Model to text transformation in practice: generating code from rich associations specifications, in: International Conference on Conceptual Modeling, Springer, 2006, pp. 63–72.

[20] J. Barnett, Introduction to scxml, in: Multimodal Interaction with W3C Standards, Springer, 2017, pp. 81–107.

[21] M. Glawe, C. Tebbe, A. Fay, K.-H. Niemann, Knowledge-based engineering of automation systems using ontologies and engineering data., in: KEOD, 2015, pp. 291–300.

[22] D. L. McGuinness, F. Van Harmelen, et al., Owl web ontology language overview, W3C recommendation 10 (2004) 2004.

[23] N. E. Fuchs, K. Kaljurand, T. Kuhn, Attempto controlled english for knowledge representation, in: Reasoning Web, Springer, 2008, pp. 104–124.

[24] R. Schwitter, K. Kaljurand, A. Cregan, C. Dolbear, G. Hart, A comparison of three controlled natural languages for owl 1.1 (2008).

[25] P. R. Smart, Controlled natural languages and the semantic web (2008).

[26] A. Crapo, A. Moitra, Toward a unified english-like representation of semantic models, data, and graph patterns for subject matter experts, IJSC 7 (2013) 215–236.

[27] S. Siemer, Exploring the apache jena framework (2019).

[28] J. Pérez, M. Arenas, C. Gutierrez, Semantics and complexity of sparql, ACM TODS 34 (2009) 1–45.

[29] L. Lim, H. Wang, M. Wang, Semantic queries by example, in: Proceedings of the 16th international conference on extending database technology, 2013, pp. 347–358.

[30] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al., Swrl: A semantic web rule language combining owl and ruleml, W3C Member submission 21 (2004) 1–31.

[31] P. Patwari, S. R. Chaudhuri, S. Natarajan,

G. Muralikrishna, M&c ml: A modeling language for monitoring and control systems, Fusion Engineering and Design 112 (2016) 761–765.

[32] D. Schnelle-Walka, S. Radomski, T. Lager, J. Barnett, D. Dahl, M. Mühlhäuser, Engineering interactive systems with scxml, in: Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems, 2014, pp. 295–296.

[33] K. Birken, Building code generators for dsls using a partial evaluator for the xtend language, in: ISoLA, Springer, 2014, pp. 407–424.

[34] Apache (2009) SCXML, howpublished = https://commons.apache.org/proper/commons-scxml/, note = Accessed: 2021-07-10, ????

[35] N. Mazalan, A. Malek, M. A. Wahid, M. Mailah, Review of control strategies employing neural network for main steam temperature control in thermal power plant, Jurnal Teknologi 66 (2014).

[36] H. QIAN, D.-j. MAO, C. LI, G.-l. HE, Alarm trigging faults diagnosis system for over-high temperature of mill outlet, Journal of Shanghai University of Electric Power (2010) 01.