TOWARDS BUILDING REUSABILITY IN CONTROL SYSTEMS – A JOURNEY

Puneet Patwari, Amar Banerjee, G. Muralikrishna, N. Swaminathan, Subhrojyoti Roy Chaudhuri
Tata Research Development and Design Centre, Pune 411013, India

Abstract

Development of similar systems leads to a strong motivation for reuse. Our involvement with three large experimental physics facilities led us to appreciate this better in the context of development of their respective monitoring and control (M&C) software. We realized that the approach to allowing reuse follows the onion skin model that is, building reusability in each layer in the solution to the problem. The same motivation led us to create a generic M&C architecture through our first collaborative effort which resulted into a fairly formal M&C domain model. The second collaboration showed us the need to have a common vocabulary that could be used across multiple systems to specify respective domain specific M&C solutions at higher levels of abstraction implemented using the generic underlying M&C engine. This resulted in our definition and creation of a domain specific language for M&C. The third collaboration leads us to imagine capturing domain knowledge using the common vocabulary which will substantially further reuse, this thought is already demonstrated through a preliminary prototype. We discuss our learnings through this journey in this paper.

INTRODUCTION

Monitoring and control systems are central to the working of projects such as SKA[1], ITER [2] and so on. These projects incorporate wide variety of heterogeneous systems and subsystems which require supervisory controllers for coordination. Our involvement with these projects gave us opportunity to learn and understand the kind of challenges involved in building monitoring and control solutions for such systems. One of our key observations is that these projects do reuse a lot of artefacts for the purpose of final implementation of their control systems. However, they still incur a huge amount of cost due to the effort they spend in the initial phases of the development life cycle. We noticed that this effort could also be substantially reduced since it showed large commonality in the type of activities taking place in each phase of their development life cycle.

Motivated by this observation, we started to analyze the prospect of a generic M&C architecture. This led to the creation of a generic M&C design and a prototype to demonstrate it in the context of GMRT. The design was inspired by the data driven paradigm and resulted in identifying a set of engines that could configure themselves based on the supplied input data that described the problem context. This approach enabled capturing the abstract model behind this input data eventually serving as the generic domain or specification model to capture the details of any M&C problem. Our first implementation realized parts of this model based on the format of the underlying execution engine which resulted into fragmentation and duplication of the M&C problem spec. This showed us the need for an integrated environment which could ensure integrity and consistency in the M&C problem specification. We recognized the need for a domain specific language (DSL) [3] to enable specification of any M&C problem so that the solutions created using the DSL could be analyzed independently. Our DSL work showed us the need for an environment which could be made aware of the application domain through its support for extensibility, analyzability, re-targetability and so on. We realized that such an environment would enable reusing a lot of domain knowledge which would enhance consistency in the entire M&C development process.

In this paper we start with a discussion on the current practice and challenges that motivated our research. Next, we highlight the proposed solutions adapted throughout our journey. Subsequently, we provide a view of our current implementation followed by the section which summarizes and concludes the paper with a futuristic view.

STANDARD PRACTICE AND CHALLENGES

Most projects start working on the requirement and design of their M&C systems from the scratch. As a result each project or groups within a project end up creating their own version of the concepts around a general problem domain such as M&C. This leads to some re-invention of concepts that are already created in another project. This point towards the lack of reusable artefacts except implementation packages across a problem domain that could enhance reusability in the entire development life cycle.

System engineering language such as SysML [4][5]provides a convenient way of expressing the designs in most of the projects. However, since much of the M&C concepts are not built into the vocabulary of SysML, it is common for different groups within projects to define the M&C vocabulary independently using SysML. Unfortunately such definitions are mostly not shared across groups. This leads to non-uniformity in the definition and usage of the M&C concepts across groups within projects. Hence it requires manual effort to

ISBN 978-3-95450-148-9

synthesize the design to create uniformity and consistency in the subsequent development phases.

The good news is, when it comes to implementation there exist a lot of reusable open source SCADA [6]packages such as EPICS [7], TANGO [8] that are popular across the scientific community.

Motivation

So it can be concluded from the discussion above, our primary motivation comes from the lack of a) reuse of domain knowledge to design M&C systems b) domain aware design analysis environment to support the development process.

PROPOSED SOLUTIONS

The course of our work can be broken into three major solution proposition milestones which are discussed below.

Inception of a Generic Architecture for M&C– SACE

Machine control systems are typically hierarchical, consisting of groups of devices at each level whose behaviour is orchestrated and coordinated by central controllers to achieve control objectives. Each device may contain actuators that perform actions on the environment, and sensors that determine the state of the environment. Thus the system can be imagined as a hierarchical network of sensors and actuators with feedback control at each level.

This led us naturally to postulate a generic architecture consisting of functionally identical nodes where the interfaces between hierarchical levels consist of three streams: commands, data and events. Each node sends commands and receives data and events from child nodes. Data and events are processed to develop the worldview. Control logic on the command path is responsible to achieve the control goals through discrete and continuous feedback control. This generic architecture was called Sensor Actuator Control Element (SACE) [9][10] and can be represented as the following figure.

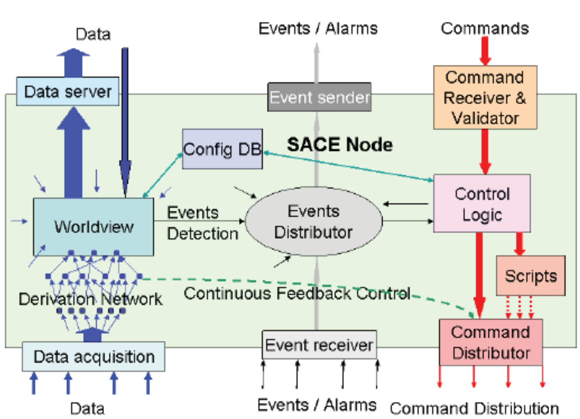

Figure 1: SACE node architecture.

The SACE node architecture incorporates the data-driven paradigm and implements each functional block using off the shelf solution components. Moreover, SACE architectural principles are driven by the requirements for efficiency, scalability and availability which are critical to machine control of scientific instruments.

The SACE prototype was initially created as a proof-of-concept inspired by the ITER architecture. This prototype was assembled by selecting freeware (and evaluation versions) of off-the-shelf technologies, and integrating them using Java. Each component was driven by configuration files, mostly in XML but in some cases had tool-specific formats. The approach of assembling off-the-shelf components made it possible to build the prototype quickly and effectively. The prototype successfully demonstrated the genericity of the architecture in the context of Giant Meterwave Radio Telescope (GMRT).

Although the prototype itself could be viewed as one more implementation of a generic M&C package, it revealed the abstract structure behind the input data which could serve as the generic domain model for M&C systems.

Creating Generic Domain Meta-model to Describe M&C Problem and Solution Design

Based on the results of the prototype discussed above, we decided to follow the Model-driven approach (MDA) [11][12]towards the definition of the M&C domain model and the creation of a domain aware environment to support its usage. This domain model would comprise of a vocabulary that can be mapped to the various concepts in the reference SACE architecture thereby establishing the correctness of the vocabulary.

The goal of this meta-modeling activity was to incorporate much of the standards from the M&C domain into the meta-model. Some elements and relations in the meta-model are custom defined due to the lack of corresponding standards e.g. terminologies such as ‘control node’ and ‘interface description’ and so on. But a good number of concepts implemented in the meta-model are borrowed from existing standards such as usage of state charts to capture behavior and so on. Since the concepts in the meta-model were derived based on the underlying modules of the SACE architecture, the meta-model also ended up becoming modular for example, concepts related to a common concern such as data acquisition generated inter relationships as opposed to the concepts belonging to a different concern such as control behavior. This also creates flexibility in making any change to a specific part of meta-model without affecting the overall structure. Figure 2 provides a glimpse of a part of the meta-model.

ISBN 978-3-95450-148-9

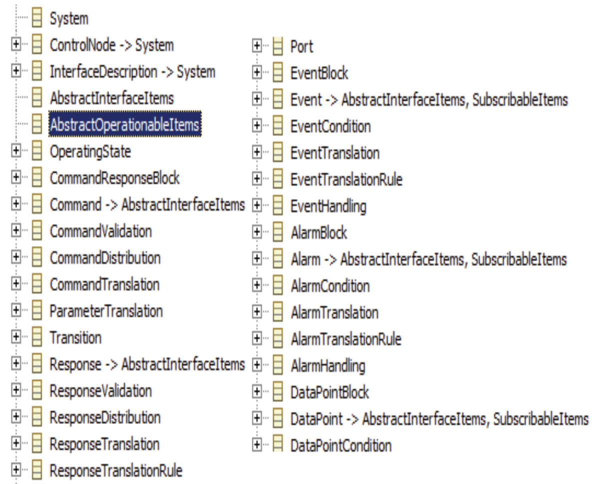

Figure 2: Meta-model: Serves as the M&C domain model.

Developing a DSL and Domain Intelligent Environment

The meta-model served as the starting point towards conceptualizing an environment that facilitates domain driven engineering to build M&C solutions. Figure 3 describes the architecture of the M&C specification environment or framework that captures an M&C problem and solution specification through the instantiation of the M&C domain model.

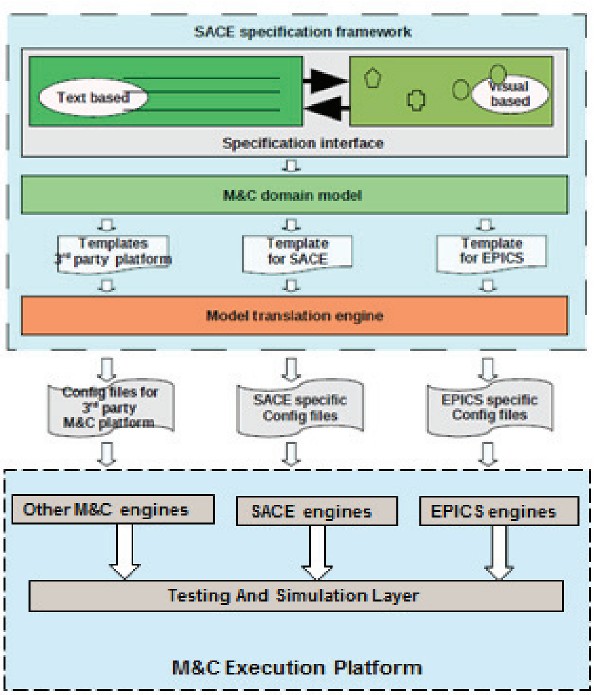

Figure 3: M&C Specification architecture.

As can be seen from the figure above, the top layer addresses the concern of a user interface to capture the M&C problem and solution specification. We implemented this layer using a DSL to capture the M&C solution details using the concepts defined by the specification model and named it M&CML [13]. The DSL helps to capture controller details such as its associated commands, responses, data streams, events, alarms, behavioral aspects like state machines, interaction with other subsystems, coordination logic and so on. The environment aims to provide support for the entire solution creation process: that is requirements, architecture, design decisions, validations and verification, creation of tests and so on. We used Eclipse based technologies such as EMF [14] [15] [16]and XText [14] [18] [18]that provide support for rapid development of DSLs following principles of MDA which suited our purpose. With XText, the various elements of the DSL along with their relationships need to be specified as grammar for the target language. Based on this, XText not only automatically generates a compiler for the target DSL but also generates along with it a complete environment to support the usage of this DSL. The environment provides built in support for user assistance, syntax highlighting, and input validation and also supports translation of the user written DSL to M&C target platform specific input formats. Since the resultant DSL environment is also a plugin in Eclipse, it allows access to other Eclipse based tools that could be leveraged for visualization or editing parts of the DSL in the future. The diagramming environment can be made complementary to the textual interface so that user could switch between textual and visual world whenever it is necessary and feasible.

The domain model is the heart of the specification framework. The domain model is implemented as a meta-model using Ecore of EMF technology. All the information captured using the DSL get populated in a meta-model instance. Hence, such specification files created independently across distributed teams can now be compared since they follow the same underlying structure. It is an effective way to bridge the gap of non-uniformity prevalent in the current approaches.

The model translation engine translates the DSL into target M&C platform specific code using model to model (M2M) transformation. This component is implemented using a combination of XTend which comes bundled with the XText tool-set and Atlas transformation language (ATL) [19] which is available as an independent plugin for Eclipse. With XTend, code generation templates can be written for each target M&C execution platform. Such code generation templates allow XTend to navigate an Ecore based M&C model instance and invoke translation logic to perform the required translation in a non-intrusive and pluggable fashion. This makes it possible for the translators to add incrementally to the framework allowing it to provide support for a wide variety range of M&C target technologies over a long period of time.

Last but not the least; the environment incorporates support to verify the solution created using this environment. Since much of the development of the controllers happen across teams in an isolated manner, the

ISBN 978-3-95450-148-9

verification of the solution becomes difficult due to the absence of the dependent systems. The testing and simulation layer of our environment provides a way to capture information about the absent systems so that their behavior could be simulated to verify the controllers which have dependency on such systems. Much of the information related to the creation of test cases could also be derived from the information present in the solution spec created using the DSL. The environment also incorporates separate meta-models to capture specific input related to testing and simulation. Based on this the environment is able to generate executable test cases and simulator implementation to verify the specified controller.

CURRENT PROTOTYPE ACTIVITY

Figure 4 shows the usage of our DSL to capture part of the M&C solution spec for GMRT.

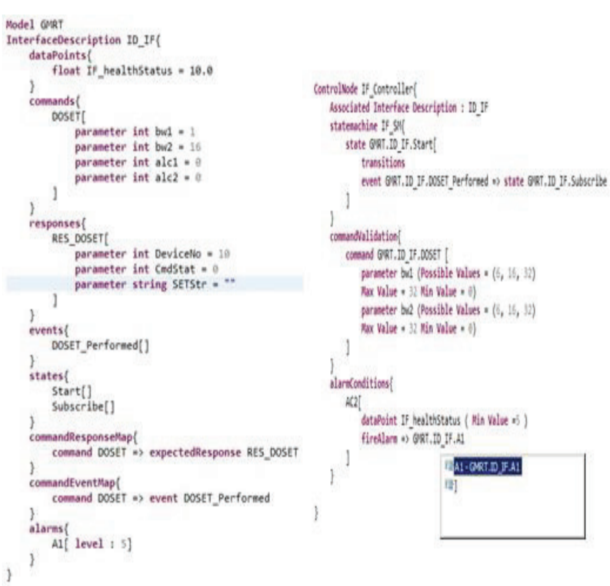

Figure 4: Sample DSL specification file.

GMRT is going through a system upgrade where they are moving from control system implemented using legacy software to a Tango based implementation. Since our environment already incorporates translators for Tango, it is envisaged that the upgradation process will get significantly augmented, since it will involve capturing the solution at a higher level and then automatically generate the implementation code specific to Tango requirements. Though, we have limited data to prove this.

Figure 5 shows a glimpse of the generated code adhering to TANGO standards and executable on the TANGO controls framework.

```
@Device
public class Controller {
    private static final Logger logger = LoggerFactory.getLogger(Controller.class);
    private static final XLogger xlogger = XLoggerFactory.getXLogger(Controller.class);
    private String className = this.getClass().getName();
    private static String deviceName = "nodes/Controller/test";
    private String parentDeviceName = "nodes/LMC/test";
    private String[] childDeviceName = {};
    /*Decialire Variables Here */

    /* Variables Declairation Ends */

     /*----------------Initialization Code--------------------*/

     @Init(lazyLoading = false)
     public final void initDevice() throws DevFailed{
        xlogger.entry();
        logger.debug("init");
@Command(name = "DOSET")
public synchronized  String DOSET(String parameters) throws DevFailed {
    System.out.println("Executing command DOSET ([mncModel.impl.SimpleTypeImpl@73a460 (name: bw1
    String inCommand = new CommandHandler().getCurrentMethod(this);
    if (new NodeCommandResponseValidation().validateCommand(inCommand,parameters)) {
String responseReceived = new ControllerSimulator().simulateResponse(inCommand, parameters);
    // Code To Be Written
    if (new NodeCommandResponseValidation().validateResponse(inCommand,responseReceived)) {
        // Calling EventDOSET_Triggered();
            return responseReceived;
    }else{
        return new String("BAD RESPONSE");
      }
    } else {
        return new String("BAD COMMAND");
    }
}
```

Figure 5: Generated TANGO specific code.

Since the environment also supports testing of the created solution, we believe this will add value to the testing and verification of the GMRT controllers significantly as well.

Figure 6 shows code for an automatically generated simulator produced by our environment and figure 7 shows a snapshot of a test run.

```
public class ControllerSimulator {
    String parseResponse = "{\"DOMON\":[{\"RES_DOMON\":[{\"allowedValues\":\"1,3,5,7,9,\
        String parseDataPoint = "{\"IF_Monitoring_HealthStatus\":{\"allowedValues\":\"10
        String deviceName = "nodes/Controller/test";
        JSONObject dataPointJsonObject = (JSONObject) JSONValue.parse(parseDataPoint);
        JSONObject jsonkk = (JSONObject) JSONValue.parse(parseResponse);
    Object object[] = dataPointJsonObject.keySet().toArray();
    public String simulateResponse(String commandName, String commandParameters) {
        JSONArray jsonArr = (JSONArray) jsonkk.get(commandName);
        if(jsonArr==null)
        {
            return "RESPONSE RECEIVED FOR "+commandName.toUpperCase();
        }
        JSONObject mlk = (JSONObject) jsonArr.get(0);

        JSONObject jso =  (JSONObject)JSONValue.parse(commandParameters);
        if (jso != null) {
        JSONObject fixedResp = (JSONObject) jso.get("fixedResponse");
        if(fixedResp!=null)
        {
        Set<String> map =  fixedResp.keySet();
        Iterator<String> iterat = map.iterator();
        String hh = fixedResp.get("Response")+":-";
        while(iterat.hasNext())
        {
            String mm = (String) iterat.next();
            if(!mm.equals("Response"))
            {
            hh= hh+ mm+":"+fixedResp.get(mm)+"||";
            }
        }
        System.out.println(hh);
        return hh;
```

Figure 6: Generated Simulator Code.

Figure 7: Generated test report.

LEARNINGS, CONCLUSION AND FUTURE WORK

Our earlier work related to the generalization of the M&C architecture now has found multiple applications for GMRT and SKA and this gives us confidence that it is possible to build reusability in the early development life cycle of M&C systems.

Although the MDE approach has been around for a while now, its application towards solving problems such as ours needs to be carefully thought through. There is always a possibility for the meta-model to become very complex very quickly. An important guidance towards the definition of the model is based on explicating concepts of the underlying architecture and seeing how much of it requires user interaction and specification. The technological support made available by framework such as XText makes is possible to build rich DSL's environments with overall support for user assistance, verification and validation and retarget-ability.

Taking this approach forward, we see the possibility of incorporating support for other aspects such as testing and verification of the developed design during the design phase itself. We foresee the possibility of explicating the application domain knowledge in an executable form so that they could be plugged into this environment incrementally providing more support towards guiding the development of M&C systems for application areas such as Radio Astronomy and so on. This could only be achieved through making this environment highly extensible and flexible which remains our endeavor.

ACKNOWLEDGMENT

We acknowledge all members of the GMRT team, members from the TM consortium and members of the ITER CODAC team for their collaboration that made it possible for us to pursue this work.

REFERENCES

[1] SKA project: https://www.skatelescope.org
[2] International Thermonuclear Experimental Reactor: https://www.iter.org
[3] Johan Den Haan, "DSL Development: 7 recommendations for Domain Specific Language design based on Domain-Driven design", http://www.theenterprisearchitect.eu/blog/2009/05/06/dsl-development-7-recommendations-for-domain-specific-language-design-based-on-domain-driven-design/
[4] Alan Moore, Sanford Friedenthal, Rick Steiner, "A Practical Guide to SysML"
[5] OMG SysML : http://www.omgsysml.org/
[6] Office of the Manager National Communication System, "Supervisory Control and Data Acquisition (SCADA) Systems". National Communications System, (October 2004).
[7] EPICS: http://www.aps.anl.gov/epics/
[8] TANGO: http://www.tango-controls.org/
[9] S. Roy Chaudhuri et al, "Integrated Monitoring and Control Specification Environment", ICALEPCS (2013)
[10] S.R.Chaudhuri, Swami N, Amrit Ahuja., "Model-driven development of control system software", in The Low Frequency Radio Universe, conference at NCRA-TIFR, Pune, Page 384(2008).
[11] Sebastien Gerard, Jean-Philippe Babau, Joel Champeau, "Model-driven engineering for distributed real-time systems", Wiley-ISTE; 1 edition, (April 5, 2010).
[12] OMG Model Driven Architecture: http://www.omg.org/mda/ (2008).
[13] P. Patwari et al, "M&C ML: A modelling language for Monitoring and Control Systems", IAEATM (2015) (Under Review)
[14] Eclipse Modeling Framework: http://www.eclipse.org/modeling/emf/
[15] F. Budinsky, D. Steinberg, R. Raymond Ellersick, E. Ed Merks, S.A. Brodsky, T.J. Grose, "Eclipse Modeling Framework", Addison Wesley (2003)
[16] Sebastian Zarnekow, Sven Efftinge: "Model Driven Software Development with Eclipse", Java User Group Hamburg, (November 2009).
[17] Lorenzo Bettini, "Implementing Domain-Specific Language with XText and XTend", (August 2013)
[18] XText: https://eclipse.org/Xtext/
[19] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I, "ATL: A model transformation tool." Science of Computer Programming 72(1/2), 31–39 (2008)

ISBN 978-3-95450-148-9