Holistic, Model-based Service Repository for Distributed Industrial Automation

Viktor Steiner, Zoltan Theisz, Zsolt Szepessy
Evopro Innovation Ltd.
Budapest, Hungary
{steiner.viktor, theisz.zoltan, szepessy.zsolt}@evopro.hu

Brice Morin
SINTEF IKT
Oslo, Norway
brice.morin@sintef.no

Gergely Mezei
Budapest Uni. of Tech. and Econ.
Department of Aut. and Applied Informatics
Budapest, Hungary
gmezei@aut.bme.hu

Abstract—Modern industrial control applications gradually take advantage of well-defined state-of-the-art distributed software architecture principles and framework designs, including corresponding modeling techniques. Moreover, the advent of this new era goes way beyond current modeling tool-chains via the application of mature meta-model driven software engineering paradigms. In the frame of the Arrowhead ARTEMIS project these techniques are to showcase the synergies between multi-level meta-model driven design and SOA based dynamic model-aware run-time framework components. This paper reports on the on-going engineering efforts in this domain by showing the generic architecture and by explaining some of the current implementation details through a prototypical tool-chain and a simple model example.

Keywords—industrial automation; SOA framework; meta-model; modeling architecture; VMTS; ThingML; SysML

I. INTRODUCTION

Collaborative automation of the Internet-of-Things era creates new kinds of services which are able to efficiently tackle major societal challenges e.g. energy reduction by smart grids and smart buildings, safer and greener transportation through Intelligent Transportation Systems, etc. These systems both rely on large-scale sensor networks and take advantage of the availability of external collaboration services for continuously sustaining acceptable and agreed on quality of service according to the negotiated SLA (Service Level Agreement). Thus, the aim of the Arrowhead project is to rationalize the notion of collaborative automation by applying and extending the concept of state-of-the-art Service-Oriented Architecture (SOA) onto Internet-of-Things solutions. This initiative goes far beyond traditional closed-loop modeling, simulation and testing approaches which are currently widespread in industrial control engineering. The complexity of the resulted distributed automation system is usually tamed by combination of an efficient service oriented run-time automation framework and the effective application of a balanced mixture of model-driven software design and traditional control engineering.

In the remainder of the paper, first in Section II, we briefly overview the current state-of-the-art modeling techniques applied for industrial control. Next, Section III explains the proposed modeling and run-time architecture of the future Arrowhead framework. Then, Section IV presents a short description of each of the incorporated modeling paradigms. Next, in Section V, the model matching bridges are showcased via a simple ThingML model. Finally, in Section VI we conclude the paper and briefly highlight our future research plans.

II. MODEL DRIVEN SERVICE AUTOMATION

Model driven engineering in the domain of mechatronics and industrial control is not a new discipline; there have been various approaches such as Intelligent Mechatronical Components [1], Automation Components [2], Model-Integrated Mechatronics [3] just to mention the most popular paradigms. All of them rely on the encapsulation of hardware and software models of the automation and control system (ACS) into individually modelled artefacts, which are stored in a central knowledge repository and hierarchically composed into the desired system. Different languages and modeling formalisms have been applied for the description of those artifacts, mostly based on particular domain specific requirements and abstraction levels, such as Matlab/Simulink, UML profiles, Labview and SysML. Although the Function Block design view enhances software reusability and provides object-oriented modularity the system design remains closed-loop. In contrary, in collaborative automation, model driven engineering is to support the design of System-of-Systems (SoS); hence it must be based, at least on the system level, on loosely coupled open-loop service oriented control principles.

III. ARROWHEAD MODELING ARCHITECTURE

The cornerstone of the Arrowhead modeling architecture is a virtually centralized, hence in its implementation distributed, service repository that supports all stages of the full life cycle of any particular service industrial automation systems may consist of. The repository ties together design-time and run-
time aspects of service models and functions via a lightweight metadata enhanced model bus. All the beneficial features of a traditional model bus[4] are kept, however bidirectional model-to-model translations are relaxed, and models and/or domain specific languages are directed by workflow orchestrated lifecycle management policies of typical Arrowhead control application developments. In practice, every modeling phase overlaps with its neighbors, mostly providing (semi-)formal information and corresponding meta-data for further elaboration towards final run-time synthesis. The proposed Arrowhead design time modeling architecture, depicted in Fig. 1, consists of three levels: System-of-Systems, System and Service.

System-of-Systems level focuses on the specification of the dynamic collaboration of the constituent systems on the global level. The modeling follows standard system engineering principles expressed in SysML. On system level the focus lies on the structural and behavioral aspects of the individual services. The system descriptions consist of both functional and non-functional elements that are constrained by the general design rules and requirements of the framework. The rules are expressed in various meta-modeling formalisms (e.g. meta-model, OCL, model verification and validation, semantic anchoring etc.) in VMTS that also include detailed and formal definition of Arrowhead’s SOA interpretation of industrial open-loop control engineering. Finally, on service level the detailed model driven engineering of control design related to the individual services is specified in various executable domain specific languages such as ThingML.

The run-time aspect of modeling framework is depicted in Fig. 2. The available hardware platforms (e.g. PLC controllers, OPC servers) implement both core Arrowhead APIs (Information Infrastructure, Information Assurance, Service Management) and support control system related functionalities such as time-synchronization. The Service Repository facilitates the global discoverability and invocation of services by coupling the run-time service database with its meta-model based model repository in VMTS.

Although some of the modeling assets, e.g. detailed FSM design, are only available as a formal non-executable specification for expert consultation, while their running implementations execute code that has been directly generated from these specification through domain specific elaboration, e.g. via ThingML, in design time.

IV. ARROWHEAD MODELING FRAMEWORK

A. SysML

The Systems Modeling Language (SysML) [5] is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and system-of-systems. Owing to these features, SysML can be used effectively in the System-of-Systems level design of the Arrowhead framework.

B. VMTS

The Visual Modeling and Transformation System (VMTS) [6] is a flexible modeling environment that also supports various model-base techniques including graph transformation, simulation, model validation and template-based model processing. VMTS is a multi-level metamodeling environment, capable of describing arbitrary domains by using its universal metamodeling technique. The environment also aids in customizing the appearance and behavior of graphical languages and also provides a text editor with syntax highlight to edit textual languages therein. Domain support in VMTS relies on generative techniques, that is, domain metamodels are processed and source code is generated from them. The generative approach makes it possible to have multiple kinds of memory representations for a domain, e.g. one with change notifications for the graphical IDE and another with high performance set for model processing purposes. The representations are completely interchangeable and they can interoperate. VMTS can handle multiple models using multiple domain-specific languages at the same time. Moreover, it is also possible to mix visual and textual languages.
C. ThingML

ThingML [7] provides support for expressing platform-independent logic, based on components and state-machines, or even via first-class action language, and it also provides a template language mechanism to implement drivers to bridge ThingML models to already existing platform-specific.

V. Model Mapping Bridges

Arrowhead’s multi-level meta-model based modeling architecture heavily relies on bilateral bi-directional mapping bridges to join the different abstraction levels, such as System-of-Systems, System and Service levels, of the contributing domain models of complex industrial control applications. In essence, the mapping bridges take advantage of the semantic overlap between the participating domain models and implement an effective and precise import-export mechanism for both-way semantically correct injection of shared model concepts. This novel mechanism has been invented to avoid the usual shortcomings originating from the establishment of a common modeling domain, which is either too overarching and thus inflexible or too restrictive and therefore uniformising in nature. Hence, our architecture design possesses only a minimal shared core meta-model based on the Arrowhead framework; however, it can easily facilitate an immense pool of bilaterally mapped, abstraction level and technical domain dependent meta-models. The required mapping bridges are implemented by taking advantage of the flexible multi-level meta-modeling and graph transformation capabilities of VMTS. The currently available mapping bridges are described in the sequel.

A. ThingML - VMTS

The ThingML to VMTS model mapping is facilitated by a hand crafted ThingML meta-model created in VMTS. The meta-model formalizes the most important structural and dynamic service specification concepts and relationships within ThingML and imports all ThingML compilable models by correspondence of those language elements to their respective meta-model peers in VMTS. The imported ThingML models may be further analyzed and validated or even transformed into other run-time service description languages available in VMTS. Due to their different model abstraction levels the ThingML meta-model in VMTS does not include all ThingML language concepts, such as expressions, actions etc. However, they are kept in their textual formats; hence also these attributes can easily be manipulated by the grammar assisted text editor facilities of VMTS. Table 1 summarizes the mapping of the most important ThingML elements onto their equivalent constructs in VMTS. Nodes and edges represent the main language concepts and relations in VMTS, while typed attributes are attached to them as labels. The attached labels play a very significant role in Arrowhead’s multi-level modeling architecture by ensuring that although the VMTS models are expressed on different abstraction levels, no information gets lost at importing ThingML models.

In general, the bidirectional extension of the ThingML-to-VMTS mapping allows smooth round-trip engineering of control applications. In practical terms, VMTS exposes a .NET based open API, which is used to create and process VMTS models from code, thus the implementation’s export-import application functions as a .NET class library. Since the concrete syntax of ThingML is textual in nature it genuinely facilitates both easy parsing and flexible generation of ThingML code. The only difficulty, however, which may occur in importing ThingML models is that their textual format is rather rigid, without the flexibilities of usual XML serializations. Therefore, not only a parser, but also a lexical analyzer was needed to be implemented during the current prototypes.

For showing the mapping, in practice, between the two model representations an example ThingML model has been selected to illustrate the operation of the import facility. The example is a simple ThingML model snippet, which describes the characteristics of a LED. While Fig. 3. shows the original ThingML model Fig. 4. depicts the result of the import process through the corresponding VMTS constructs of the mapped model.

<table>
<thead>
<tr>
<th>ThingML element</th>
<th>VMTS construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thing</td>
<td>Node</td>
</tr>
<tr>
<td>Port</td>
<td>Node</td>
</tr>
<tr>
<td>Message</td>
<td>Node</td>
</tr>
<tr>
<td>State</td>
<td>Node</td>
</tr>
<tr>
<td>Action</td>
<td>Node</td>
</tr>
<tr>
<td>Action body</td>
<td>Textual attribute</td>
</tr>
<tr>
<td>Expression</td>
<td>Node</td>
</tr>
<tr>
<td>Expression body</td>
<td>Textual attribute</td>
</tr>
<tr>
<td>Event</td>
<td>Node</td>
</tr>
<tr>
<td>Concrete Event (Receive Message)</td>
<td>Textual attribute</td>
</tr>
</tbody>
</table>

TABLE I. MAPPING OF THINGML ELEMENTS INTO VMTS CONSTRUCTS

![Fig 3. Input Model in ThingML](image)

Fig 3. Input Model in ThingML.
The semantic bridging between SysML and VMTS relies on a tailor made SysML meta-model in VMTS, similar to the case already discussed in the previous paragraph describing the ThingML-VMTS bridging. However, here only those parts of SysML that are mandatory in the design phases of the Arrowhead framework are fully modelled, that is, the current selection consists of the following: Block Definition Diagram, Internal Block Diagram, Use Case Diagram, Requirements Diagram, and a simplified version of Sequence Diagram.

For the implementation of the prototype, Sparx Systems’ Enterprise Architect [8] was used as our preferred SysML modeling environment due to the fact the modeling scenarios made available by most of the Arrowhead partners were specified in this tool. Enterprise Architect exports models into OMG standard XMI format, which was parsed and imported into VMTS without major difficulties, therefore, this part of the export-import facility is currently fully operational. However, the exporting of those VMTS models that Enterprise Architect can import is not a trivial task at all. Indeed, in order to implement this direction, EA’s method of generating type and element identifiers need to be replicated, which can be done fully only in direct cooperation with Sparx Systems.

All in all, although our proof-of-concept implementation works well the export-import application for SysML models is currently work in progress and the final tool selection is also under investigation in order to choose the most appropriate SysML modeling environment that is both beneficial and acceptable for the majority of Arrowhead partners.

VI. CONCLUSION

Our paper has described the generic modeling architecture and the currently selected elements of the modeling tool-chain of the first prototypical implementation of the Arrowhead meta-model driven holistic model mapping infrastructure. Although the current implementation is rather limited in scope and supports only a single executable modeling framework, which is ThingML, it is generic in design and highly modular and plug-in friendly in nature. Hence, various other executable models will get integrated into the framework in the coming phases of the Arrowhead project following those architectural and implementation principles that have been invented and successfully validated during this first prototype. The fully functional modeling system will be part of the Arrowhead software asset and its details will be published in further industrial and research papers.

ACKNOWLEDGMENT

This work is supported by the EU Artemis project “Arrowhead”.

REFERENCES