Towards a Solution for Synchronizing Disparate Models of Ultra-Large-Scale Systems

James H. Hill, Jules White, Sean Eade, Douglas C. Schmidt
Vanderbilt University
Nashville, TN, USA
{j.hill, jules.white, sean.eade, d.schmidt}@vanderbilt.edu

Trip Denton
Lockheed Martin, Advanced Technology Lab
Cherry Hill, NJ, USA
ldenton@atl.lmco.com

ABSTRACT

Traditional model-driven engineering (MDE) techniques rely on a paradigm where systems are developed using tightly coupled, monolithic modeling tools. Such monolithic modeling tools address many concerns, but operate largely in isolation of one another. As system size and complexity grow to become ultra-large-scale (ULS) systems, it is becoming clear that no single monolithic modeling tool can capture all the concerns of an ULS system. It is therefore essential that isolated modeling tools collaborate with each other when realizing ULS systems.

This position paper presents our approach to facilitate collaboration between disparate MDE tools and their models. Our approach is based on model attributes, which are key/shared assumptions/-concerns about an ULS system, extracted from a source model and used to synchronize disparate models. Our approach is suitable for ULS systems because the independent relation created between the isolated models and the model attributes enables independent trade-off analysis between models, decentralized development of models, and integration with inconsistent and rapidly changing models that are ideal for a particular domain or feature of a ULS system.

Keywords

continuous model integration, model-driven engineering, model synchronization, ULS systems

1. INTRODUCTION

Key challenges of model-driven ULS system development. Traditional model-driven engineering (MDE) [14] has shown great promise when building medium- to large-scale systems [1,11]. MDE helps raise the level of abstraction of system design and allows developers to express their intent and work with artifacts that are more closely related to domain constructs than third-generation programming languages. MDE also alleviates many of the inherit complexities associated with building large-scale systems, such as documenting design specifications [12], verifying functional properties [2], validating non-functional properties [6], or solving deployment & configuration (D&C) problems [15].

As systems grow larger and more complex to become ultra-large-scale (ULS) systems [7], however, a single MDE technique or tool (such as domain-specific modeling languages [10] or formal models [3]) is insufficient to provide all the required support. Different system concerns, such as its fault-tolerance capabilities, real-time schedulability, and software-to-hardware deployment topology, require different languages to precisely analyze the system [9]. It is hard to leverage conventional MDE techniques for ULS systems if (1) each modeling language or tool is used in isolation due to dependencies between models and (2) decisions in one model have unforeseen consequences in other models.

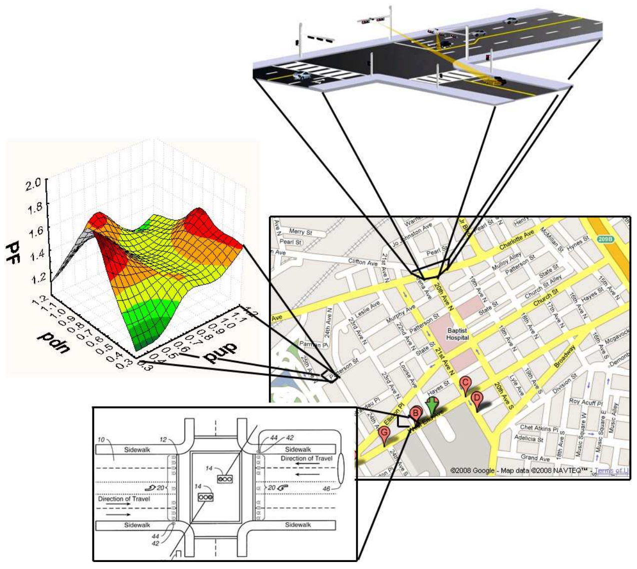

Figure 1: Intelligent Transportation System

For example, an intelligent transportation system [4], as shown in Figure 1, that coordinates its operations with many intersections in a city may have a UML model of the system's conceptual implementation (*e.g.*, classes, sequence, and use-case diagrams), a formal model to verify functional properties (*e.g.*, deadlock and state reachability), and a system execution model to validate non-functional properties (*e.g.*, worst-case system execution time). Moreover, different portions of the system may be developed by different groups dispersed throughout a region, which implies different—possibly conflicting—underlying concerns and assumptions of the system under development. If system developers want to leverage a new model (such as a D&C modeling language) or implement new functionality (such as evaluating the effect of checkpointing the system's state), it is imperative that the ULS system's models collaborate to ensure each addresses their specific problem with the same underlying system assumptions and remain appropriately consistent with one another.

Maintaining consistency between models is necessary even for small-scale systems because a single model of a system is rarely sufficient to model all relevant aspects of a system. For small-scale systems it is feasible to maintain this consistency manually. Such an approach is problematic for ULS systems, however, since they are created by many developers, working in different organizations, distributed across many regions and domains, using multiple disparate MDE techniques and tools. When developers maintain consistency between these different modeling boundaries manually they often make assumptions about the ULS system to map the conceptual model to a concrete model that fits within their span of interest/responsibility.

For instance, in our intelligent transportation system example the UML model and the system execution model may have different assumptions about how checkpointing is implemented, or the formal model and system execution model may differ in their checkpointing frequency assumptions. These different assumptions also will affect how the D&C model deploys the realized ULS system. In particular, these assumptions create diverging and inconsistent solutions between models that need to collaborate to realize a working ULS system, such as the intelligent transportation system illustrated in Figure 1.

Solution approach → Model synchronization via model interfaces and attributes. To address the problem of collaboration and synchronization between models of ULS systems, developers need new techniques that will allow disparate MDE techniques and tools to communicate seamlessly when creating and deploying ULS systems. This paper describes our approach enabling synchronization between disparate models of ULS systems.

Our approach uses *model attributes*, which are key/shared assumptions/concerns about an ULS system, *model interfaces* and *connectors*, which are used to described and insert/extract the model attributes into/from their target/source model, respectively. Our approach also allows the seamless integration of new models (*i.e.*, model plug-and-play) so they can collaborate with existing disparate models of the ULS system. Our initial observations show that this approach enables independent trade-off analysis between models, decentralized development of models, and integration with inconsistent and rapidly changing that are ideal for a particular domain, or feature, of a ULS system.

Paper organization. The remainder of this paper is organized as follows: Section 2 elaborates our approach to model synchronization for ULS systems; Section 3 describes initial results realized by our approach; Section 4 compares our approach with related work; and Section 5 presents concluding remarks and future research directions.

2. SYNCHRONIZING DISPARATE MODELS OF ULS SYSTEMS

In Section 1, we discussed the challenges of synchronizing disparate models of ULS systems. To address these challenges, a methodology is needed that allows disparate models—which can be dispersed widely throughout regions—to exchange common knowledge, such as functional (*e.g.*, checkpointing frequency), implementation (*e.g.*, portions of the systems affected by checkpointing implementation) and deployment (*e.g.*, target hardware/software) requirements. Such a methodology should provide the following features:

- A database that contains a disjoint subset of *model attributes*, which are system properties that must be shared between multiple disparate models and stored in a well-defined format, such as a scalar value, comma-separated values, or verbose XML, of the realized ULS system. Model attributes represent the minimal information needed to ensure disparate models maintain consistent assumptions based on executing the source model, *i.e.*, evaluating it based on its current values. Moreover, the model attributes help to prevent diverging solutions, similar to an invariant specified in formal model checking [2].
- *Model interfaces*, which describe the input/output model attributes for a particular model type, and *model connectors*, which are implementations of a model interface and understand how to read/write a subset of model attributes needed to maintain a consistent view of the ULS system. Individual models read/write the model attributes to/from their target database via model connectors. Since multiple databases may need to store a disjoint subset of the ULS system's properties that must be shared between disparate models, model connectors are responsible for resolving the location(s) of the model attributes. Model connectors are also bound to a particular model type to promote reuse across multiple domains and solutions.
- Generic and extensible *plug-and-play* support for modeling languages and tools that is not bound to a particular format, language, or specification. As the ULS system evolves, new/-different models will be added to their current design space. Such models will also begin to read/write their own model attributes. A plug-and-play framework enables support of future and unknown modeling languages because they only have to describe their model interface and provide a model connector to begin read/writing model attributes. By make the plug-and-play framework extensible, different modeling tools and languages can be used to specialize the existing infrastructure without breaking it. For example, a new modeling tool integrated into the framework may specialize it to validate the value of an attribute using a XML schema definition without requiring the other modeling tools and languages to implement the same functionality.

Our solution approach is shown in Figure 2. All model attributes

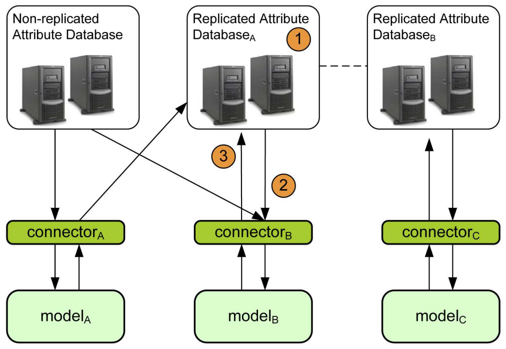
Non-replicated Attribute Database
Replicated Attribute Database$_A$
Replicated Attribute Database$_B$
1
2
3
connector$_A$
connector$_B$
connector$_C$
model$_A$
model$_B$
model$_C$

Figure 2: Conceptual Overview of ULS System Model Synchronization

(1) are stored in well-defined location(s), such as a database or repository. Due the scale of the system, it is possible to replicate attribute database(s) as shown in Figure 2 so models use an appropriate database, *e.g.*, one closest to their location. Each attribute

database contains a disjoint subset of properties, such as properties for a specific version, concern, or feature, of the ULS system under development. When developers need to update their model (2) they use the model's corresponding connector to read the appropriate model attributes, which ensures the local working copy of the model makes assumptions consistent with those of other models.

Figure 2 also shows that model connector's write model attributes back to the attribute database. After developers finish updating the local working copy of their model, *e.g.*, evaluating their model based on the new/updated ULS system assumptions, they use the model connector (3) to write their model attributes back to the appropriate (replicated) database. Although this process could take some time to converge due to ULS system scale, it ensures that all models continue to maintain a unified view of the ULS system consistent with the changes made.

3. INITIAL OBSERVATIONS OF MODEL SYNCHRONIZATION

Section 2 described our solution approach to enable disparate models to coordinate with one another in ULS systems. This section presents some initial observations of using model attributes and connectors to facilitate model synchronization.

Support for loose coupling of modeling tools/environments/-languages. Model attributes are pushed/pulled to/from the model, respectively, via model connectors. Since model connectors are responsible for handling model attributes, the actual models (*i.e.*, those within a developers local workspace) are not concerned with the format of the actual model attribute. This approach creates a loose coupling between disparate modeling tools/environments/-languages, such as those illustrated in Figure 3, and allows them to remain independent from each other—similar to how the *Bridge* and *Adapter* pattern [5] allow two unrelated objects to collaborate without becoming tightly coupled.

Due to the loose coupling, the disparate models can collaborate without becoming tightly coupled to other models. Moreover, we can integrate (*i.e.*, plug-and-play) new tools/environments/languages that address specific concerns of the ULS system as needed, such as the D&C modeling language for the intelligent transportation system shown in Figure 3, without breaking existing models of the ULS system.

Trade-off analysis of ULS system properties. Model attributes are assumptions about the ULS system's properties between multiple disparate models. Before system developers use one or more models, they must be updated with the latest properties from the model attribute database (*e.g*, the attribute repository in Figure 3). Once the models are updated, they can be evaluated to understand how the updated assumptions affect the current model, thereby enabling system developers to conduct trade-off analysis on key ULS system properties, such as understanding how different checkpointing frequencies affect end-to-end worst case execution time. Such trade-off analysis properties can be written to the model attribute database via the model connectors to preserve the learned facts about the ULS system, *e.g.*, the intelligent transportation system in Figure 3, and used by other disparate models.

Partial knowledge of model (and system) to achieve synchronization. A challenge of using disparate models is maintaining consistent information between them. In many cases, one model may only need a fraction of the information in another model, *i.e.*, a partial view of the system, to maintain a consistent view of the entire system. By using model attributes we alleviate the complexity of needing complete knowledge of a model (or the system) to synchronize disparate models. Since each model synchronizes itself based on its desired model attributes, which are specified via model connectors, it only reads a subset of the model attributes stored across all attribute databases to maintain a consistent view of the system.

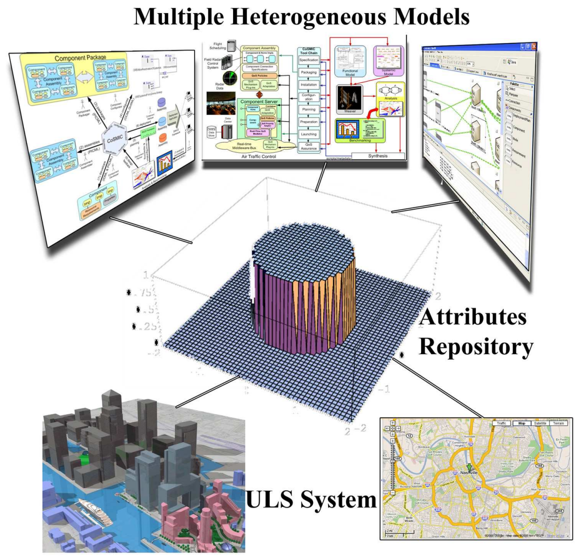

Figure 3: Model Synchronization for Intelligent Transportation System

Automation of the model synchronization process. Each model interface determines what model attributes to read/written to/from the attribute repository. When the model interfaces and connectors and the actual models are stored in a well-known location for integration, such as a repository—and we assume that no two models can write the same model attribute—we can use topological sort [13] to build a dependency graph between each model based on their model attributes. This graph determines the required order that we must evaluate each disparate model to produce output model attributes needed as input model attributes for other models. By leveraging this capability, it should be possible to automatically synchronize disparate models, ensure they maintain a consistent view, and validate them based on the common assumptions specified by model attributes.

Support for decentralized MDE. As discussed in Section 2, model attributes are stored in a database, which could be replicated. These attributes can be spread across multiple locations, where each location stores a disjoint subset of the ULS system's model attributes. The model connectors, however, are responsible for resolving the actual location of a model attribute. Based on our model attributes and model connectors, we provide a decentralized MDE-based approach to synchronizing disparate models.

4. RELATED WORK

This section compares our work on synchronizing the assumptions between different models of a ULS system with related work on model synchronization. Prior work on modeling has largely focused on small-scale systems where a single model or tool is sufficient. Since ULS systems do not fit this single model/tool mold, we do not compare with these existing monolithic modeling techniques.

Zave et. al [16] describe techniques for collaborating between disparate models in the domain of formal specification and verification of programs. Their solution mapped all models to a common simplified predicate logic—similar to the MetaObject Facility for domain-specific modeling languages. Although this approach is valid, it means all disparate models have complete knowledge of the entire program, which is not feasible for ULS system models because they span many domains. Our approach differs from Zave et. al because we do not map the model attributes to a common representation. Mapping model attributes to a common representation is particularly hard when disparate models of ULS systems have overlapping concerns, but disjoint semantics, purposes, and underlying formalisms. Moreover, our approach alleviates the need for disparate models to have complete knowledge of the system and focuses on attributes (or assumptions) that are necessary for it to solve its problem.

The (Web-based) Open Tool Integration Framework (OTIF) [8] is a tool that provides collaboration between disparate models. OTIF's uses graph transformations and rewriting techniques to transform models between isolated tools, which is more of a point-to-point solution. Our solution approach is different in that we do not perform model transformations and rewriting techniques to achieve model synchronization. Instead, we make model attributes, which are common assumptions about the system, the primary artifacts for synchronizing disparate models. Moreover, OTIF's solution implies that models have a complete view of the system and are tightly coupled; whereas, our solution approach implies that models have a partial view of the system, *i.e.*, the minimal knowledge necessary to synchronize disparate models, and are loosely coupled.

5. CONCLUDING REMARKS

As ULS systems become more prevalent, multiple models will be needed to express different system design concerns. To ensure that each model has a consistent view of the system's assumptions, disparate models will need to exchange information. This paper described our solution approach to enable disparate models to collaborate, which is based on storing common assumptions about the system in *model attributes* and using *model interfaces* and *connectors* to manage model attributes for individual models. Our initial results indicate that this approach enables disparate models to collaborate without needing complete knowledge of the entire system.

The following list summarizes our future research directions for enabling model synchronization between disparate models in ULS systems:

- Automatically maintaining consistency between models should ideally occur continuously throughout the development lifecycle of the system. Our future work therefore involves understanding the benefits of using continuous integration environments to enable the continuous model integration.
- When synchronizing many disparate models there will be times when different models will have conflicting results or assumptions based on the evaluation of their subset of model attributes. Our future work therefore involves understanding how to locate such problems and how to resolve them both autonomously and manually.
- There can be use cases where the dependencies between model attributes form a cyclic graph, such as a feedback loop between two models. Our future work therefore includes understanding how to handle such use cases to prevent the synchronization process from entering infinite loops.

Acknowledgements

This work was funded in part by Lockheed Martin Advanced Technology Labs in Cherry Hill, NJ. We would particularly like to thank Rick Buskens, Edward Jones, and Srini Srinivasan for their help on this project.

6. REFERENCES

[1] K. Balasubramanian, A. Gokhale, J. Sztipanovits, G. Karsai, and S. Neema. Developing Applications Using Model-Driven Design Environments. *IEEE Computer*, 39(2):33–40, Feb. 2006.

[2] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In *SFM*, pages 200–236, 2004.

[3] M. Chechik and A. Wong. Formal Modeling in a Commercial Setting: A Case Study. *Journal of Systems and Software*, 60(1):59–82, 2002.

[4] S. Fritsch, A. Senart, D. C. Schmidt, and S. Clarke. Time-bounded Adaptation for Automotive System Software. In *Proceedings of the Experience Track on Automotive Systems at the 30th International Conference on Software Engineering*, Leipzig, Germany, May 2008.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[6] J. H. Hill, J. Slaby, S. Baker, and D. C. Schmidt. Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS. In *Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, August 2006.

[7] S. E. Institute. Ultra-Large-Scale Systems: Software Challenge of the Future. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, Jun 2006.

[8] G. Karsai, A. Lang, and S. Neema. Design Patterns for Open Tool Integration. *Software and Systems Modeling (SoSym)*, 4(2):157–170, 2005.

[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, pages 220–242, June 1997.

[10] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, pages 44–51, November 2001.

[11] G. Madl and S. Abdelwahed. Model-based analysis of distributed real-time embedded system composition. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 371–374, New York, NY, USA, 2005. ACM Press.

[12] Object Management Group. *Unified Modeling Language (UML) v1.4*, OMG Document formal/2001-09-67 edition, Sept. 2001.

[13] D. J. Pearce and P. H. J. Kelly. A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs. *Journal of Experimental Algorithmics (JEA)*, 11:1.7, 2006.

[14] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

[15] J. White, D. C. Schmidt, and A. Gokhale. Simplifying autonomic enterprise java bean applications via model-driven development: a case study. *Journal of Software and System Modeling*, 2007.

[16] P. Zave and M. Jackson. Where Do Operations Come From?: A Multiparadigm Specification Technique. *IEEE Transactions on Software Engineering*, 22(7):508–528, July 1996.