Advancing Software Architecture Modeling for Large Scale Heterogeneous Systems

Ian Gorton and Yan Liu
Pacific Northwest National Laboratory,
Richland, WA
{ian.gorton,yan.liu}@pnl.gov

ABSTRACT
In this paper we describe how incorporating technology-specific modeling at the architecture level can help reduce risks and produce better designs for large, heterogeneous software applications. We draw an analogy with established modeling approaches in scientific domains, using groundwater modeling as an example, to help illustrate gaps in current software architecture modeling approaches. We then describe the advances in modeling, analysis and tooling that are required to bring sophisticated modeling and development methods within reach of software architects.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architectures;* D.2.2 [**Software Engineering**]: Design Tools and Techniques – Evolutionary prototyping;

General Term
Design

Keywords
Software architecture, modeling, dependability

1. INTRODUCTION
In the last two decades, software architecture has become foundational in the development of large, heterogenous software intensive systems. The critical nature of software architecture is well understood both in industry and research. Software architecture pervades all phases of software development and is the key to being able to sensibly evolve a system over its lifetime [15].

Much of the progress in software architecture can be broadly described as adhering to the principle of explicitly capturing architecture design decisions, whereby design artifacts can be expressed and communicated by human-beings, and codified and processed by machines. This has led to a profusion of proposed architecture design notations, architecture styles, and architecture description languages such as xADL[9], UML, and model driven frameworks [3,12], to name but a few.

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

In mainstream software architecture practices, modeling drives the specifications of components, connectors and their overall interactions to constitute a software architecture. Designs are typically partitioned into various views that abstractly describe artifacts such as system structure, behavior and deployment. Together, the collection of views represents the logical system architecture, and is used to guide subsequent development, often in a highly iterative fashion.

Large scale, complex systems that drive businesses, large scale enterprises and Internet-facing applications are invariably comprised of a variety of off-the-shelf software platforms and technologies such as middleware, web technologies and databases. These infrastructural technologies provide facilities for building application components and connecting them to the network as services. They have been developed over many years and provide robust, scalable components that can be tailored to satisfy a specific application's requirements. For these reasons, these technologies are intrinsic in delivering the dependability attributes of a system. However, state-of-the-art software architecture techniques basically assume that the specific features and capabilities of these infrastructural technologies can be abstracted so that a uniform architecture description language can express the design in a manner that is agonistic to specific technologies being considered for a system.

Abstracting away the specific quality attributes that are intrinsically supplied by underlying software platforms is a major problem in architecture design. While at design time, application-specific components can be logically divorced from the underlying infrastructures, at run-time this does not hold. Hence application run-time qualities such as performance, scalability and reliability must be considered as a totality of the application's components, both application-specific and infrastructural. Empirical studies such as [1] clearly demonstrate this, showing the considerable variability in an application's performance based on the selection of Java application server technology.

The desirability of being able to analyze an architecture design early and often in a project lifecycle has been recognized by others. Accordingly, mainstream software architecture modeling approaches have been augmented in numerous research projects in attempts to capture various quality concerns. These include:

1. Annotations or profiles [2] to enrich the expressiveness of the models. These permit quality attribute-relevant details to be specified as properties of components or connectors;
2. Techniques and tools for transforming annotated architecture models [12] into formats that can be either simulated or solved using analytical models. The aim of automated

analysis is to produce predictions of how the application may perform under "what-if" scenario analysis;

3. Methods and mechanisms for building formal models into an application so that the application behavior is guided to adapt to changing conditions in the supporting infrastructure and in the surrounding physical environment [4, 5];
4. Frameworks to predictably construct assemblies of components, given components with properties that can be objectively measured or verified by independent parties [6].

Despite all this progress, large scale software system architecture design remains elusively in the realms of experts. State-of-the-art software architecture research methods and tools have, with a few notable exceptions typically in narrow application domains [7], failed to gain traction in practice.

In this paper we argue that advances are needed to both increase the capability and practicality of software architecture methods and tools. Practitioners need accurate and defensible prediction capabilities for the quality attributes of their designs before any critical design decisions and investments are made. Another requirement is to evaluate the uncertainty of the predictions, as this can be used to quantify design risks.

Invariably, reasoning about complex systems requires the use of models. However it seems unlikely that further incremental advances in existing modeling approaches focused on the level of component assemblies can scale for systems that are implemented using different computing models, programming languages, or integrated across technology boundaries. Hence, we see the need for new theories, methods and tools that can scale up to the scope of large scale systems. Such systems are characterized by having multiple stakeholders, being developed over extended time phases, and built over legacy systems to accommodate new features. They are highly complex, distributed, and often connect heterogeneous subsystems across organizational boundaries. New methods and tool are needed to make it possible to model and analyze architecture designs for such systems. Keeping these models synchronized and relevant to the application implementation is also crucial for long-lived applications.

2. A MODELING ANALOGY

In the design of a large-scale, heterogeneous software system, system architects aim to translate quality attribute requirements into a design that can potentially satisfy these requirements. During the design process, architects are faced with multiple dimensions of uncertainty about the dependability attributes that a given solution can provide. Abstract models (e.g. UML) are used to describe the key design elements, and in standard industrial practice, the models simply serve the purpose of documentation and communication of ideas.

From these designs, architects informally reason about the dependability attributes of the proposed solution. Typically in cases where there appears to be significant risks (e.g. performance/reliability of a component), the design guides prototyping exercises that implement key parts of the design and validate it against requirements. In this sense, prototyping collects concrete data about key component characteristics that is used to evaluate and reduce the uncertainties associated with the dependability of the proposed architecture.

Consequently, software architects expend significant amounts of effort in modeling and validation of their designs. However the models cannot formally exploit concrete component execution data from prototypes to give deeper insights into the application design through simulation or analytical techniques. For complex, long lived software systems, this strikes us as unsatisfactory and unlikely to lead to major breakthroughs in producing higher quality systems at lower costs.

In many scientific fields, the approaches taken contrast markedly with software architecture modeling. As an example, consider Figure 1, which overviews how geoscientists model and simulate flows of containments in subsurface groundwater flows.

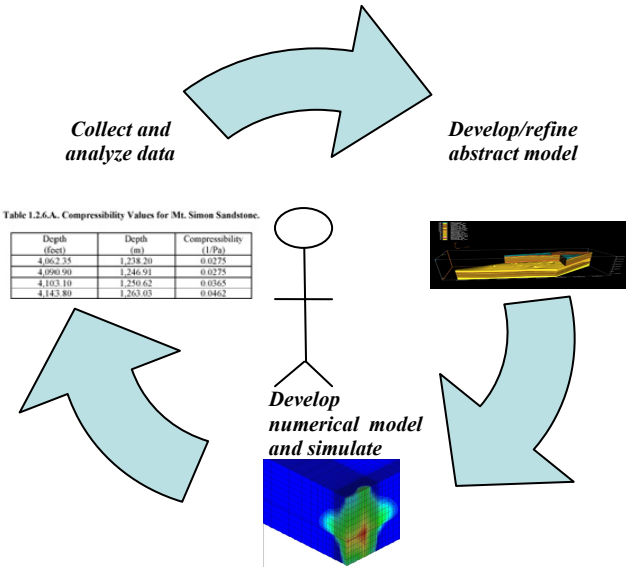

Figure 1 Iterative modeling workflow for geoscientists

Geoscientists initially build 3 dimensional models of the subsurface geology. These represent the layers of rock types in a given area, their extents and properties, and other geological features such as faults. These models are based on any available geological data for the site being modeled, and require considerable expertise to create. As it is not possible to exactly measure the geology of the site (we can't 'see' underground), geoscientists leverage their knowledge of geology and mathematical methods to estimate as best they can the structure of the site. Next, they map this abstract site model to a numerical grid, provide a set of initial conditions (e.g. river flows, daily rainfall), and use a simulator to show how contaminants move through the groundwater over a period of many years. Importantly, as new data about the site geology is gathered from field surveys and experiments, the abstract model is refined and new simulations are performed to improve the accuracy of the results.

Geoscientists also recognize that their model predictions are almost always incorrect, as the models can never completely accurately represent the subsurface geology and chemistry. For this reason, they employ well known mathematical techniques and frameworks to estimate realistic parameter ranges for their models based on concrete data, and quantify uncertainty in their results using for example Monte Carlo simulations.

The above is a very simplified description of groundwater modeling, but many analogies can be drawn with software architecture design. For example:

- Both create abstract models of an artifact of interest

- Both rely on deep levels of experience and knowledge from the foundations of their scientific discipline
- Both rely on concrete data to inform the evolution of the abstract model

However, the major difference lies in the refinement of abstract models into models that can be analyzed, and the recognition that imperfect model results require quantification of uncertainty. In Software engineering, while model-driven development approaches can generate technology-specific executables from models, the level of model description required is far too detailed for early architecture design activities. Unlike geoscientist and scientists in other disciplines who use modeling and simulation extensively, software architecture lacks a systematic approach for refining abstract models into forms that can be analyzed for attributes such as performance, scalability and reliability. We believe that without such approaches, large scale applications will continue to run unacceptably high levels of risks due to our lack of understanding of the influence of architecture design decisions on project quality, schedule and cost.

3. WHAT IS NEEDED

We see a need to conduct new research to bridge the gulf between abstract architectural models and current and future MDD approaches. We envision architecture toolsets that are able to continuously integrate, manage and visualize the connections between architecture models. The toolsets would also manage multiple data sets obtained from tests and benchmarks on the technologies of interest for a project, and enable architects to construct and analyze technology-specific models of their designs using a variety of simulators or mathematical solvers.

depicts the major artifacts and transformations that we envisage. Architects would initially construct an abstract model that is used in the project exploration phase to explore design options and communicate with stakeholders. When the initial design stabilizes, the architect can incorporate technology-specific component models into their design, and parameterize these models with data representing their performance, scalability and reliability. This data may be obtained from prototypes created by the project team, or from data catalogues that give measures for a component based on standard benchmarks. Finally, the architect defines the initial conditions for the model, such as request arrival rates, data sizes and constraints such as connection or thread limits. This creates a model that can be fed into a simulator or analysis tool to produce predictions about the application's dependability attributes.

Concurrently, techniques are needed to map the abstract model into a Platform Independent Model (PIM) that becomes the input to an existing MDD tool chain. This is then transformed using standard MDD meta-models and tools into a Platform Specific Model (PSM) by incorporating details of the target infrastructure technologies. From this model, code can be generated to potentially validate results from analytical models or simulations, or to commence detailed development.

The linkages between models would be preserved by parameters representing architecture choices, domain specific application behavior, technology characteristics, and infrastructure computing capacities. This would permit on-going, multi level model manipulation to verify the dependability attributes, costs, and other factors against the system requirements.

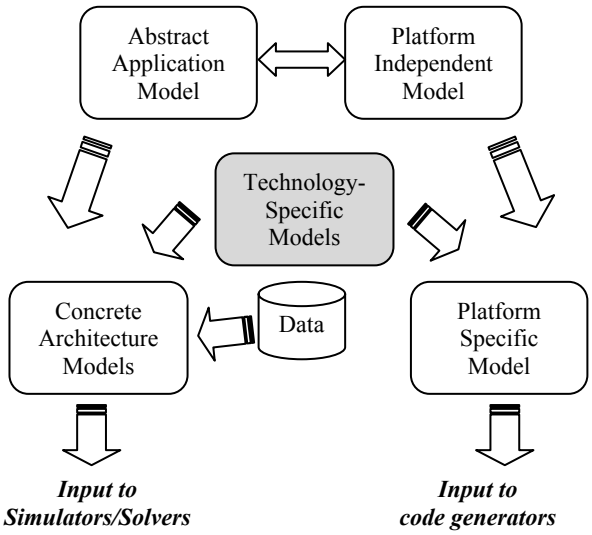

Figure 2 Advanced Architecture Analysis Toolset

To achieve these goals, we envisage a research agenda incorporating the following:

Building technology specific models. Akin to geology site modeling to predict groundwater movement, any useful software architecture prediction model needs to capture the execution environment supplied by infrastructural software technologies. Current approaches tend to model infrastructures monolithically, and are forced to make simplifying assumptions about the internals of the software (e.g. threading, disk accesses). This is not sufficient to accurately predict detailed behaviors. A new approach would leverage specific modeling approaches (such as mathematical formulas, Petri-nets, stochastic processes, networked graphs, or metrics in spreadsheets) for the individual components in the infrastructure platforms. These complex technologies comprise layers of composable services for handling concurrency, enforcing security policies, routing messages, data access, and so on. Each service has its own properties and constraints that influence the architecture design, and hence must be abstracted into separate models described by parameters. For example, a Java Messaging Service (JMS) can be modeled as a queuing element with parameters characterizing its message processing rate, maximum queue size, and overheads incurred by transactional messaging protocols.

Two different state-of-the-art practices may be exploited to build these technology specific models. One exploits annotation and profiling approaches [13] whereby a metamodel captures the key entities in the technology specific models which are required for building analytical or simulation models. Accordingly the abstract application model is annotated by profiles conforming to the metamodel. The transformation between the abstract application model and the technology specific model is via the immediate metamodel. This approach generally relies on complex model transformation tools and metamodeling notations such as MOF.

Alternatively, Domain Specific Language (DSL) could be developed for individual technologies leveraging frameworks such as Microsoft's DSL or the Eclipse EMF. Key entities in the technology specific models become first class modeling elements,

available for the application modeling. As a result, the abstract application models and technology specific models can converge into the same design artifact. Since the models are processed by a DSL engine, it is possible to inform the DSL engine of how to invoke the executable model solvers or the simulation tools from within the design. The coupling between DSL and technology features however, raises the challenge of making the DSL extensible to incorporate changes as the technologies evolve.

Building concrete architecture models. A concrete model is derived from the technology specific model reflecting architecture design options. Design options have dependencies on the configuration settings of the infrastructure supplied by a technology and application components. These different settings (such as stateful vs. stateless components) affect the behavior of the infrastructure [8], and their effects can be absorbed into random variables with specified distributions. These settings are modeled as properties, constraints or functions. Different input values for these settings can be explored in "what-if" scenarios for the concrete architecture models to predict quality attributes.

Calibrating models with profiles and benchmarks. The above capabilities produce predictions for the designed system in the form of a model with parameters relating to the specific infrastructure technology. Some of the parameters represent tunable features of the container's configuration such as thread pool size, but others reflect internal hidden implementation details of the container, which may not be measurable directly. Therefore the solution requires running benchmarks to estimate the values of parameters. Since benchmark measures can be compared to model predictions, we can solve the parameterized model and determine the values of the missing parameters. This approach is equivalent to how geoscientists estimate parameter values for their models.

Benchmark scenarios must be carefully designed to exercise the key elements of a specific infrastructure (including software and hardware capacities) involved in the system design. In a manner compatible with our ideas in this paper, research efforts (such as [10,11,16]) have already demonstrated the utility of MDD tools to automate benchmark generation and measurement collection. The raw data collected may be further filtered and aggregated before input into the model. As this procedure may requires heuristic inputs to guide the data characterization and the solution of the model, it remains an intriguing question to determine the extent that this procedure can be automated and integrated with architecture design/analysis tool chains.

Quantifying prediction uncertainty. Individual model predictions will almost never be correct, because so many factors can introduce errors in the modeling process These include measurement errors, infrastructure variability, assumptions on application behavior, and so on. In science, well understood methods for quantifying model prediction uncertainty exist[14]. These basically generate ensembles of parameter and model variations for execution, analyzing model outputs to determine key sources of uncertainty and developing strategies for efficiently reducing uncertainty. Incorporating these methods and tools in the architecture modeling tool chain is necessary to allow architects to effectively understand and quantify the risks in their designs. Early research experience [17] has been reported to scope uncertainties in domain models akin to the intrinsic models in our proposed analysis toolset (see Figure 2), namely *Abstract Application Model* and *Platform Independent Model*. Since the source of uncertainties may spread several models in the analysis toolkit, it remains a challenging issue to quantify the uncertainties at individual models.

4. CONCLUSIONS

Several initial efforts from groups worldwide and ourselves give us confidence that this vision of an integrated, modeling-based approach to software architecture design is scientifically feasible. Interestingly, technological trends too are converging to make the problem tractable. Modeling of specific technologies and attempting to parameterize them for an effectively infinite range of deployment scenarios seems a massive challenge. However, the emergence of Cloud computing, where a small number of core hardware and software platforms are available for running applications, potentially simplifies the challenge immensely. Such trends make us optimistic that, for the many projects that cost $10s to $100s of millions to develop, more rigorous software architecture modeling practices can one day become ingrained in practice and provide architects with a deeper understanding of their design options at all stages in a project lifecycle.

5. REFERENCES

[1] Gorton, I., Liu, A., and Brebner, P. 2003. Rigorous Evaluation of COTS Middleware Technology. *Computer* 36, 3 (Mar. 2003), 50-55.

[2] Woodside, M., Franks, G., and Petriu, D. C. 2007. The Future of Software Performance Engineering. In *2007 Future of Software Engineering* (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 171-187.

[3] Douglas C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," Computer, vol. 39, no. 2, pp. 25-31, Feb. 2006.

[4] Zhang, J. and Cheng, B. H. 2006. Model-based development of dynamically adaptive software. In *Proceedings of the 28th international Conference on Software Engineering*. ICSE '06. ACM, New York, NY, 371-380.

[5] Kramer, J. and Magee, J. 2007. Self-Managed Systems: an Architectural Challenge. In *2007 Future of Software Engineering* (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 259-268.

[6] Wallnau, K. 2003. Volume III: A Technology for Predictable Assembly from Certifiable Components, Technical Report CMU/SEI-2003-TR-009, Software Engineering Institute.

[7] Ommering,R., Linden, F., Kramer, J., Magee, J., "The Koala Component Model for Consumer Electronics Software," Computer, vol. 33, no. 3, pp. 78-85, Mar. 2000.

[8] Liu, Y., Fekete, A., and Gorton, I. 2005. Design-Level Performance Prediction of Component-Based Applications. *IEEE Trans. Softw. Eng.* 31, 11 (Nov. 2005), 928-941.

[9] Dashofy, E. M., Hoek, A. V., and Taylor, R. N. 2001. A Highly-Extensible, XML-Based Architecture Description Language. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture* (August 28 - 31, 2001). WICSA. IEEE Computer Society, Washington, DC, 103.

[10] Grundy, J., Cai, Y., and Liu, A. 2005. SoftArch/MTE: Generating Distributed System Test-Beds from High-Level

Software Architecture Descriptions. *Automated Software Engg.* 12, 1 (Jan. 2005), 5-39.

[11] Zhu, L., Bui, N.B., Liu, Y., Gorton, I., MDABench: Customized benchmark generation using MDA, Journal of Systems and Software, Volume 80, Issue 2, February 2007, Pages 265-282, ISSN 0164-1212.

[12] Becker, S., Koziolek, H., and Reussner, R. 2009. The Palladio component model for model-driven performance prediction. *J. Syst. Softw.* 82, 1 (Jan. 2009), 3-22.

[13] D. B. Petriu and M. Woodside, "A metamodel for generating performance models from uml designs," in *7th International Conference on Modelling Languages and Applications, Lisbon, Portugal, October 11-15*, ser. LNCS, T. Baar, A. Strohmeier, A. Moreira, and S. J. Mellor, Eds. Springer Berlin / Heidelberg, 2004, vol. 3273, pp. 41-53.

[14] S. F. Wojtkiewicz , M. S. Eldred , R. V. Field , A. Urbina, J. R. Red-horse, Uncertainty Quantification In Large Computational Engineering Models, In Proceedings of the 42rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, number AIAA-2001-1455, 2001.

[15] Taylor, R. N. and van der Hoek, A. 2007. Software Design and Architecture The once and future focus of software engineering. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 226-243.

[16] Bošković, M. and Hasselbring, W. 2009. Model Driven Performance Measurement and Assessment with MoDePeMART. In Proceedings of the 12th international Conference on Model Driven Engineering Languages and Systems (Denver, CO, October 04 - 09, 2009). A. Schürr and B. Selic, Eds. Lecture Notes In Computer Science, vol. 5795. Springer-Verlag, Berlin, Heidelberg, 62-76.

[17] Betty H. C. Cheng, Pete Sawyer, Nelly Bencomo and Jon Whittle, A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty, MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS, Lecture Notes in Computer Science, 2009, Volume 5795/2009, 468-483.