Position Papers of the Federated Conference on Computer Science and Information Systems pp. 93–97
DOI: 10.15439/2018F375
ISSN 2300-5963 ACSIS, Vol. 16

Simulation Driven Development of Distributed Systems – Coupling of virtual and real system components

Tommy Baumann
Andato GmbH & Co. KG
Ehrenbergstraße 11
98693 Ilmenau, Germany
tommy.baumann@andato.com

Bernd Pfitzinger
Toll Collect GmbH
Linkstraße 4
10785 Berlin, Germany
bernd.pfitzinger@toll-collect.de

***Abstract*—Looking at the end-to-end processing, typical software-intensive systems are built as a system-of-systems where each sub-system specializes according to both the business and technology perspective. One challenge is the integration of all systems into a single system – crossing technological and organizational boundaries as well as functional domains. To facilitate the successful integration, we apply the system design process Simulation Driven Development (SDD). The basic idea is the application of realistic simulation models in parallel to existing software engineering procedures to enable testing and validation from day one. In this article we discuss the coupling of sub-systems implemented as virtual simulation models with sub-systems implemented as real sub-systems to enable unit tests against system level. Two approaches are presented: loose coupling and tight coupling.**

I. Introduction

THE ever-increasing complexity of software-intensive systems is an ongoing challenge to the tools and methods used in system engineering [1]. The building blocks of software-intensive systems or even systems-of-systems are often already systems themselves – either as commercial-off-the-shelf software used with 'minor' adaptations or custom-made software. This reuse of existing and proven solutions allows for the rapid construction of large systems. In the system development process, the engineering practices typically address – successfully – the scope of a single, independent system: The requirements are (and can be) defined, the system is modeled, implemented, tested and put into operations.

What started as the 'waterfall model' [2] has been adapted to mitigate the risks of failing to deliver a system in time, in budget and in quality. The software engineering models address the risk of failure in several ways: Using formal rather than natural language reduces misunderstandings; a small project scope reduces the 'value at risk' and testing assures that the intended level of quality is reached. These different approaches can be seen in a software engineering meta-model, like the V-Model XT [3] [4], a model in widespread use in Germany. The formal, document-oriented waterfall model is represented by the left side of the V, proceeding through the typical phases towards the implementation. In particular the model requires for each phase to define a dedicated test phase at a later time (the right-hand side of the 'V'). Depending on the focus of a particular project the meta-model allows for modern approaches e.g. test-driven development [5] or agile methods [6].

Yet methods proven to deliver successful implementations at the level of a single system seem to be inadequate for large-scale systems. [7] argues that implicit assumptions – the development process can be controlled, decisions are rational, the problem can be defined and delimited – break down when independent systems interact. Often the behavior of a coalition of systems is emergent and it is difficult or impossible to predict. An unintended consequence could be the complete failure of the overall system without a clear software bug within any contributing system ([8] gives the 2010 stock market flash crash as a prominent example).

We propose extending common software engineering practices through the use of simulation models. At any stage during the system development process the current level of detail needs to be captured in form of an executable specification. The system design process SDD follows exactly this axiom [9]. A major challenge of SDD is the coupling of different simulation models, simulators and already implemented system components to enable unit tests within the expected operational context. Each requirement and their test cases depend on the context, i.e. reusing a component or integrating a system into another system will most probably invalidate many assumptions and test cases [10]. Features, performance and operational aspects need to be proven before a sub-system is ready for integration. The development of such simulation-based test environments is a challenging task: sub-systems can be interfaced with simulation models where the remaining sub-systems operate at a higher degree of abstraction and the technical performance of the simulators can be insufficient.

In this article we discuss the possibilities and technical challenges regarding the coupling of simulation models with real world implementations. In section two we first introduce the SDD approach and show how modeling and simulation technologies can be applied to leverage specification quality and speed in the development of complex distributed systems. Section three explains the idea of loose coupling

and applies it in the example of the German toll system. In section four we explain the idea of tight coupling and apply it on the example of testing an electronic control unit (ECU) of a car. In the last section, we summarize and provide an outlook about future work.

II. Simulation Driven Development

Simulation Driven Development (SDD) is an efficient methodology for developing complex distributed systems. In contrast to conventional methodologies simulation models of the system to be developed are used during the entire development process [11]. These simulation models allow description of static and dynamic system properties which enormously improves specification quality and speed. Furthermore, they are used for analysis, evaluation, validation, verification and optimization during the course of the developing process. In addition to the simulation model of the system to be developed, SDD uses a simulation model of the development process itself. This simulation model allows automation of development steps in the form of workflows like optimization cycles, test cycles, revision controls and document sharing with component manufacturers. Fig. 1 shows SDD in the form of an extended V-Model.

User requirements of the system to be developed are determined and transferred into an initial simulation model during the requirement analysis. It represents the dynamic user behavior as well as the user interaction with the system in form of application scenarios / use cases. The simulation model is the basis for future acceptance tests and allows the first analyses in developing the design of the system.

In the subsequent phase of system specification, developer requirements (functional and not-functional) are derived and transferred into an executable specification. This is a simulation model of the whole system encompassing technical functions, architecture and user behavior. Each element of the simulation model is related to the user's or developer's requirements to ensure their compliance. The simulation model allows an early validation of the whole system against the application scenarios and allows locating integration problems (dynamic coupling effects between system components). Furthermore, it permits system-level optimizations and is the basis for later system integration tests.

Within the component specification each system component is further refined in form of simulation models. Starting point is the present executable system specification with the component related requirements and parameters. Simultaneously the executable system specification serves as development environment for component validation and optimization against the holistic system. As a consequence of the refinement, the components and thus also the system parameters become more specific. The component specification forms the basis for the later unit tests and component integration tests.

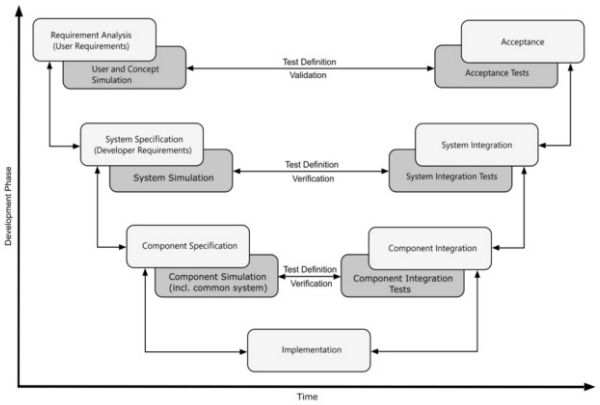

Fig. 1: Simulation Driven Development as a V-Model

After the specification of the system and its components the development process proceeds with the implementation. Based on the previously specified test cases the implementation will be verified and validated (performance tests and endurance tests). First, component integration tests are performed. In this step the functionality of each component is viewed in isolation. Afterwards components are integrated to a system followed by system integration tests. Finally, the system is tested against the initially defined user requirements by an acceptance test. SDD allows the reuse of the previously defined test cases which exist in form of simulation models. To that extent, simulation models are attached to the system implementation by Hardware/Software in the Loop (SiL/HiL).

To implement SDD a formal definition of the overall process is required. Figure 1 depicts the SDD approach as a V process model enriched by simulation-based test activities throughout the whole process. As in the classical V process model the level of detail increases when going downwards in the process and it is possible to go back to upstream process steps if it emerges that changes are required. The result of each process step is captured by one or more simulation models which provide test drivers for integration steps like "in-the-loop", integration or even acceptance tests. It is therefore similar to the W model in software development [12]. Both models share the concept of "testing starts at day one", i.e. that tests performed during the design stages prepare the integration tests and that test teams and system designers work together from the very beginning. A key improvement is that SDD features dynamic testing by the use of simulation models from the very beginning which directly supports dynamic testing in the integration phase in contrast to the classical W model where only static analysis is performed at early design phases.

Test failures or emerging new requirements can lead to changes in upstream (more general) designs which in turn might invalidate downstream (more detailed) specifications and simulation models. It is therefore critical to maximize the overall degree of formalism, automation and requirements traceability within and between each of the

steps in the SDD process model. This in turn requires detailed workflows for which individual working steps can then eventually be executed automatically by a workflow engine.

As mentioned above, SDD follows the idea of "test starts at day one". A prerequisite is the presence of an executable system model throughout all design stages. To enable unit tests, the system models need to be coupled to existing hardware or software components (HiL and SiL) which can be implemented in two different ways: loose coupling and tight coupling [13] [14]. While loose coupling doesn't consider the dynamic interaction between the sub-systems, tight coupling does.

III. Loose Coupling

Loose coupling simplifies the implementation of unit tests of relatively independent sub-systems. The prerequisite is a mostly unidirectional exchange of data. e.g. lacking feedback or disregarding feedback that deviates from the simulation. This enables technically simpler test benches, since a sub-system can be tested without real-time connectivity to the rest of the system. One possible implementation is to simply take the events passing through the simulation model and export those events that are relevant to a particular domain-specific interface. The event list is a flat file with a chronological list of points in time and additional domain specific information.

In the example of the German automatic toll system we use this approach to provide the temporal behavior of 1.1 million on-board units communicating with the central system [15]. The event list contains all TCP/IP connections and their domain specific payload, e.g. the number of journey data files transmitted. Operational testing takes this temporal data, adds the domain specific content and uses another more technical simulator to drive the real-world data center applications. The example illustrates two challenges encountered in this approach:

- The abstraction gap – when a high-level simulation model encounters a real-world system – is overcome by additional simulation models. Loose coupling of the simulation models immediately translates into loose coupling of the technical teams involved.
- The simulation performance – where a real-world system must not be impacted by the simulation performance. Loose coupling allows to precompute at least parts of an operational test.

IV. Tight Coupling

Tight coupling is used to develop and test sub-systems that work closely together, e.g. as parts of an embedded system. The entire logic of such systems is typically distributed across several sub-systems. These sub-systems need to be connected dynamically to deliver the desired logic/process. Each sub-system is characterized by interconnectivity and inter-processing to the other sub-systems and depends strongly on them.

Two possibilities of technical implementation can be distinguished: direct tight coupling using data files, shared memories or network protocols and indirect tight coupling using a central data distribution service (DDS). Fig. 2 shows direct coupling using TCP/IP sockets. Each participating simulator needs to implement an adapter to each connected simulator – a time-consuming and expensive task. All requirements regarding quality of service (real-time capabilities, distribution of simulators across multiple nodes), standardization of data types (different data types of simulators) and time synchronization (global clock as simulation time) need to be considered. These issues are solved using a central data management entity like DDS.

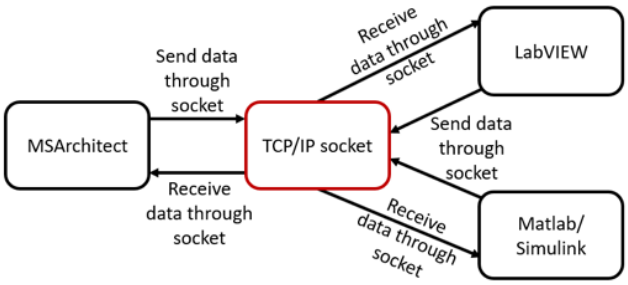

Fig. 2: Direct tight coupling using TCP/IP

DDS is a networking middleware that simplifies complex network programming and distribution of data. It implements a publish–subscribe pattern for sending and receiving data, events, and commands among the nodes. Nodes that produce information (publishers) create "topics" (e.g., temperature or pressure) and publish "samples". DDS delivers the samples to subscribers that declare an interest in that topic. DDS handles transfer chores: message addressing, data marshalling and de-marshalling (so subscribers can be on different platforms from the publisher), delivery, flow control, retries, etc. Any node can be a publisher, subscriber, or both simultaneously.

The DDS publish-subscribe model virtually eliminates complex network programming for distributed applications. The key benefit is that applications that use DDS for their communications are decoupled. Little design time needs be spent on handling their mutual interactions. In particular, the applications never need information about the other participating applications, including their existence or locations [16] [17]. Fig. 3 shows the basic architecture of a DDS solution.

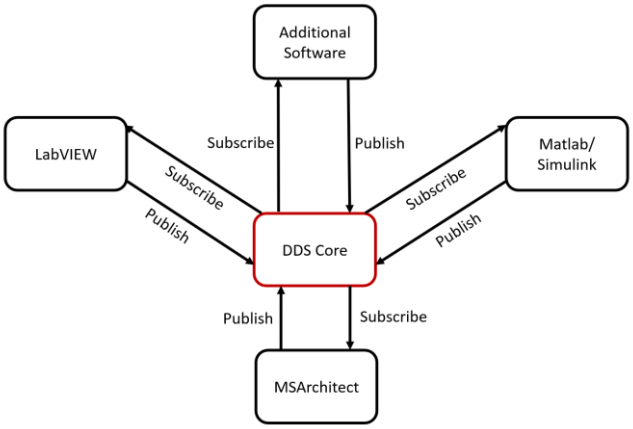

Fig. 3: Indirect tight coupling using DDS

We have developed an ECU test environment which couples three different simulation tools (MSArchitect, Matlab/Simulink and LabVIEW). It is used to validate and test the function and performance of the interface between the three simulation tools. The coupling between those systems is accomplished by using TCP/IP sockets achieving a high data transfer rate.

In this example MSArchitect takes the server role while Matlab/Simulink and LabVIEW are the clients. During the initialization phase of the simulation the server is waiting for incoming connections. Once all clients have been connected to the TCP/IP sockets the simulation starts.

The tight coupling example caused several challenges which have to be solved. The data transfer through TCP/IP sockets is limited to character arrays. Therefore, the different data types of the several simulation tools could cause issues because of the conversion in character arrays and the conversion back to the data types. For this reason, a definition of supported data types and a standardization of those data types need to be developed. Furthermore, the time of the connected simulators need to be synchronized, for example by implementing a global clock, controlled and distributed by one simulator.

Fig. 4 shows the user interfaces of the test environment. The whole car is modelled in MSArchitect and parts of the model are implemented in LabVIEW or Matlab/Simulink. The LabVIEW model contains the entire logic of the ECU which is built in the car model. Matlab/Simulink provides the logic and behavior of the combustion engine. The performance of the ECU test environment is limited by the hardware resources of the test computer.

In our case the desired maximum performance of the implemented interface could not be accomplished since all three models were running on the same system. In future work, we want to split the simulation models over three processing nodes to reach a significant higher performance of the entire simulation. At this point, the ECU test environment is running without any data loss or synchronization issues, which in fact is a great starting point for further development.

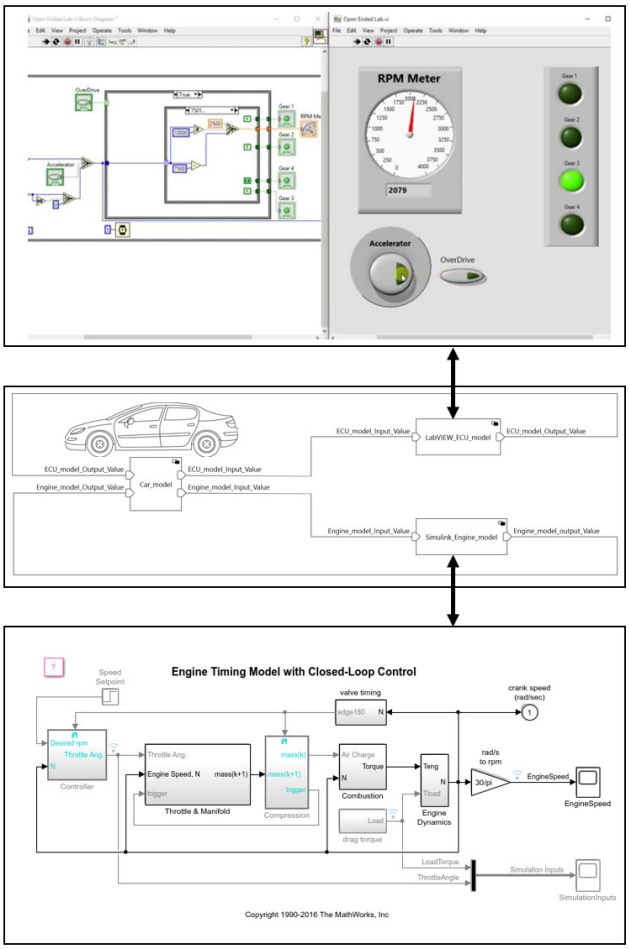

Fig. 4: ECU test environment

V. Conclusion

We have shown the applicability of simulation models along the development process using a simulation driven design process. From the onset the simulation model provides the operational context and over time the level-of-detail can be increased to the point where real-world sub-systems interface with the simulation model. In the article we discussed how simulation models can interface with real-world hardware or software to enable realistic unit tests. Two possibilities have been introduced: loose coupling and tight coupling.

Next, we're planning to extend our DDS solution and connect the examples mentioned above to this solution. Above all, securing a defined quality of service within DDS will be a challenging task. In addition, we're preparing a guideline to select a proper coupling mechanism.

References

[1] B. Boehm: A view of 20th and 21st century software engineering. Proceedings of the 28th international conference on Software engineering, ACM, 2006, pp. 12-29. doi: 10.1145/1134285.1134288.

[2] W. W. Royce: Managing the development of large software systems. Proceedings of IEEE WESCON, Los Angeles, Aug. 1970, pp. 1-9.

[3] Verein zur Weiterentwicklung des V-Modell XT e.V. (Weit e.V.): V-Modell XT version 2.0. [accessed 21-Jan-16]. Available: http://www.v-modell-xt.de/

[4] S. Biffl, D. Winkler, R. Höhn, and H. Wetzel: Software process improvement in Europe: Potential of the new V- Modell XT and research issues. Software Process: Improvement and Practice, vol. 11, no. 3, pp. 229-238, Jun. 2006. doi: 10.1002/spip.266

[5] D. Janzen and H. Saiedian: Test-Driven Development: Concepts, taxonomy, and future direction. Computer, vol. 38, no. 9, pp. 43-50, Sep. 2005. doi: 10.1109/MC.2005.314

[6] Manifesto for agile software development, [accessed 29- Jul-2015]. Available: http://www.agilemanifesto.org/.

[7] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. McDermid, and R. Paige: Large-scale complex IT systems. Communications of the ACM, vol. 55, no. 7, pp. 71-77, Jul. 2012, doi: 10.1145/2209249.2209268

[8] D. Cliff and L. Northrop: The global financial markets: An ultra-large-scale systems perspective. In R. Calinescu and D. Garlan (Eds.): Large-Scale Complex IT Systems. Development, Operation and Management. Ser. Lecture Notes in Computer Science, vol. 7539, pp. 29-70, Berlin: Springer, 2012. doi: 10.1007/978-3-642-34059-8_2.

[9] Baumann, T, “Simulation-driven design of distributed systems”. In SAE International, SAE Technical Paper, pp. 1-7, 2011. doi:10.4271/2011-01-0458.

[10] E. J. Weyuker: Testing component-based software: A cautionary tale. IEEE Software, vol. 15, no. 5, pp. 54-59, Sep. 1998. doi:10.1109/52.714817

[11] B. Pfitzinger, T. Baumann, and T. Jestädt, “Simulation driven development - validation of requirements in the early design stages of complex systems - the example of the German toll system," in Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, M. Ganzha, L. Maciaszek, and M. Paprzycki, Eds., ser. Annals of Computer Science and Information Systems, vol. 11, IEEE, Sep. 2017, pp. 1127-1134. doi: 10.15439/2017F133.

[12] Gerrard, P. and N. Thompson: „Risk-Based E-Business Testing“, Artech House INC 2002, ISBN-13: 9781580533140, ISBN-10: 1580533140

[13] A. Salkintzis, C. Fors, and R. Pazhyannur, "WLAN-GPRS Integration for Next-generation Mobile Data Networks," IEEE Wireless Communications, vol. 9, no. 5, pp. 112-124, October 2002. doi: 10.1109/MWC.2002.1043861

[14] Beck, F. and Diehl, S., “On the Congruence of Modularity and Code Coupling”, in 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference (ESEC/FSE '11), Szeged, Hungary, September 2011. doi:10.1145/2025113.2025162

[15] Pfitzinger, B.; Baumann, T.; Jestädt, T.: Network Resource Usage of the German Toll System: Lessons from a Realistic Simulation Model. In: 46th Hawaii International Conference on System Sciences (HICSS) (2013), pp. 5115–5122. doi: 10.1109/HICSS.2013.41

[16] Rekik, R. and Hasnaoui, S., “Application of a CAN BUS Transport for DDS Middleware”; 2009 Second International Conference on the Applications of Digital Information and Web Technologies, London, 2009, pp. 766-771.doi: 10.1109/ICADIWT.2009.5273919

[17] Deniz, E. et al., “DDS Based MIL-STD-1553B Data Bus Interface Simulation”; The Journal of Defense Modeling and Simulation, vol 12, issue 2, pp. 179 – 188. doi: 10.1177/1548512914530534