Abstract

UML2 and SysML try to adopt techniques known from software development to systems engineering. However, the focus has been put on modeling aspects until now and quantitative performance analysis is not adequately taken into account in early design stages of the system. In this paper, we present our approach for formal and simulation based performance analysis of systems specified with UML2/SysML. The basis of our analysis approach is the detection of communication that synchronize the control flow of the corresponding instances of the system and make the relationship explicit. Using this knowledge, we are able to determine a global timing behavior and violations of this effected by preset constraints. Hence, it is also possible to detect potential conflicts on shared communication resources if a specification of the target architecture is given. With these information it is possible to evaluate system models at an early design stage.

1. Introduction

Even though the complexity of complete systems on chips is steadily increasing, their demanded time-to-market is shortening. New methods for the handling of the complexity need to be evolved. The Unified Modeling Language (UML) [11] is a general-purpose modeling language widely used in the software development process for the specification and the design of software systems. Modeling systems with UML allows to start modeling at a high abstraction level and the description of aspects like timing or architecture even before parts of the functionality are specified. The requirements in the area of electronic system design, especially in the area of systems-on-chip, demand for the quantitative evaluation of different design alternatives. The application of UML in the system level design process demands for an early exploration of designs, the verification of constraints and the recognition of violations of the specification. These requirements are currently not covered. The earlier design decisions can be verified, the earlier conflicts and violations of constraints can be recognized. Subsequent changes in the configuration often need a redesign of the complete architecture. Requirements like response times or throughput have to be considered and conflicts on shared resources have to be identified and eliminated. Figure 1 presents an overview of our approach for the analysis of UML/SysML models. We do not require a complete functional and architectural specification of the system. We need to define three properties of the system: the architecture, the functionality in terms of the control flow and timing constraints. UML provides different ways for the specification of the control flow like state machines, sequence diagrams, collaboration diagrams and activity diagrams. The specification of the duration and latency of processing and communication can be done e.g. by time-annotated sequence diagrams or by timing diagrams. Although our approach is able to consider combinations of them, we will focus on sequence diagrams for the definition of control flow and timing in this paper. The ability to specify combined fragments like loops and alternative statements in UML2 sequence diagrams provides a flexible framework for the modeling of the control flow between messages.

An architectural model is used to describe the mapping of activities and communication to components. Furthermore, we apply it for the description of constraints like latencies of communication resources and for the description of properties like shared media usage. We use UML structured classes/SysML assemblies [16] for the specification of the architecture of the system. We extract the semantics of the functional and architectural specification and transform them to a formal functional model and to an architectural meta model. The combination of both represents a system model we start our analysis from.

Figure 1. Analysis flow

An architectural model is used to describe the mapping of activities and communication to components. Furthermore, we apply it for the description of constraints like latencies of communication resources and for the description of properties like shared media usage. We use UML structured classes/SysML assemblies [16] for the specification of the architecture of the system. We extract the semantics of the functional and architectural specification and transform them to a formal functional model and to an architectural meta model. The combination of both represents a system model we start our analysis from.

Our formal analysis approach of the control flow and timing behavior allows the estimation of communication schedules and...
conflicts caused by overlaps in the schedules. Furthermore, we are able to determine performance parameters like best and worst case response time (BCRT, WCRT) of the whole system and guaranteed I/O data rates the system is able to handle.

The assumption is that the instances in the model will be realized either as specialized hardware or as processors or microcontrollers executing software threads. Depending on the realization and the partitioning of the instances, the communication will be handled as IPC between threads or they will be mapped to communication resources like on-chip busses, on-chip networks or dedicated wires. Our analysis approach relies on the impact of blocking communication on the runtime of the whole system. We developed a method for formal performance analysis to determine the influence of blocking communication on the timing behavior of the system. Therefore the presented control flow and the timing information are transformed to a communication dependency graph (CDG). This kind of representation has a more formal semantics than the original format. The runtime of blocks between the communication events can be determined by execution time analysis of an existing behavioral description [4, 2] or by user annotations originating from experiences, assumptions or the specification. The combination of UML/SysML specifications with analytical results of existing IP and libraries leads to structured reuse of qualified IP in platform based design. Our method is not restricted in use with the complete description of the system. Often some typical use cases, which embody the critical situations by the meaning of timing or communication of the envisaged system are modeled with sequence diagrams. The validation of these scenarios helps to detect presumable bottlenecks from the developers point of view without having to model the complete behavior of all parts. The previously determined results will be used to evaluate predefined architectural alternatives, decide if all constraints are met and if the design complies with the system specification.

Furthermore, the system model can be transformed to an executable and efficient simulation model. This allows us to couple the specification with existing systems and to monitor properties in parallel to the analysis. The combination of the performance estimation and the evaluation of specified architectures at an early design stage results in a very valuable approach in the area of SoC design.

In the next section, we shortly present related work in the area of performance analysis in UML. Section 3 describes our approach to extract the semantics of sequence diagrams, to formalize the notation by reducing the amount of different used expressions. The extraction of the meta information from the architectural model is briefly described in Section 4. Section 5 describes our method for communication analysis. The application of the results of the communication analysis for formal performance and conflict analysis is described in Section 6. The aspect of a simulation model is covered in Section 7 briefly. An example is presented in Section 8.

2. Related Work

The application of UML in the design process tries to provide a structured view and an exchange mechanism. Several approaches aim towards a formalization of the UML to avoid discrepancies between developers and to provide better consistency between the different representation styles of UML [6, 3, 7]. The different approaches can be initially distinguished by the elements of the UML standard they are examining. Sequence diagrams and state machines are widely used in the analysis process. There are methods of performance engineering distributed software systems [15, 9]. On account of their targeted application domain, conflicts on shared resources and other hardware related issues are not detected by these approaches. Other analytical models analyze time-enhanced UML diagrams with the aid of stochastic processes [8, 5, 1, 18]. Although it delivers good results in application domains like mobile communication where mean values are sufficient, we demand for worst-case estimations in the area of real-time systems-on-chip and distributed embedded systems. Relating to this paper, there has been work towards communication analysis we are setting up our concept [14]. The scope in these works concerns the analysis of implemented systems, we are focusing the design stage in contrast.

3. Transformation of Sequence Diagrams

Sequence diagrams allow a graphical representation of interactions and causal relations between instances in the system. The temporal behavior is covered by the UML-SPT profile [10]. The ability to specify constraints like maximum durations and characteristics as execution times and communication delays enables the description of various temporal behavior and dependencies. Although they are easy to understand, the notation is due to the arbitrary expressions not as formal as needed to be directly analyzed with regard to performance aspects. The approach presented in this section demonstrates the transformation of sequence diagrams to our analysis model, the communication dependency graph (CDG). A CDG denotes a consolidated representation of communication and the temporal and causal behavior between them. A communication dependency graph can be defined as follows: A CDG is denoted by [2]

\[
CDG := (V_{CDG}, E_{CDG}, E_{COM}, \tau_{CDG}, l_{CDG}),
\]

where

- \( V_{CDG} \) is a set of nodes representing communication nodes or loops with unbounded data dependent delay,
- \( E_{CDG} \subseteq V_{CDG} \times V_{CDG} \) is a set of directed edges describing the precedence dependencies between nodes,
- \( E_{COM} \subseteq V_{send} \times V_{rec} \) with \( V_{send} = v \in V_{CDG}: TCDG(v) \in \{ \text{sendasync}, \text{sendasync} \} \) and \( V_{rec} = v \in V_{CDG}: TCDG(v) \in \{ \text{receivasync}, \text{receivasync} \} \) is a set of directed edges describing the communication
- The function \( TCDG(v) : V_{CDG} \rightarrow \{ \text{sendasync}, \text{sendasync}, \text{receivasync}, \text{receivasync} \} \) denotes the type of each node.
- The edge weights are represented by the function \( l_{CDG} : E_{CDG} \rightarrow N \cup \{ 0 \} \) with minimum and maximum execution time \( l_{CDG}(v_1, v_2) = (c_{min}, c_{max}) \) between the nodes \( v_1, v_2 \in V_{CDG} \).

Figure 2 presents an example of a CDG consisting of two communicating processes.

The first step for the construction of the CDG is the identification of communication events as \textit{send} and \textit{receive} in the system described with sequence diagrams. These events embody the nodes of the CDG. Obviously, this information can be directly adopted from the sequence diagrams. The paired nodes from each communication are connected by communication edges in the CDG. The dependencies between the nodes are derived from the control flow between communication events.
if the lifeline of the instance in the sequence diagram connects both events directly without passing any other communication event.

**Figure 2. Communication Dependency Graph**

The investigated representation style can contain several new expressions since version 2 of the UML standard, like loops, alternative and optional statements, that influence the control flow. Hence, these statements have to be transformed context and semantics preserving to the analysis model. We propose a recursive algorithm to break up interlocked combined fragments. For preserving the semantics, we introduce extensions to the CDG. For a more compact representation of loops in the control flow, the specification of the CDG is extended by

- The function \( \sigma_{CDG}(e) \) : \( E_{CDG} \rightarrow N_0 \times N_0 \) denotes the minimum and maximum numbers of the execution of a back pointing edge in the control flow.
- The function \( \rho_{CDG}(e) \) : \( E_{CDG} \rightarrow N_0 \) denotes the execution order of an edge. For \( e_1, e_2 \in E_{CDG} \) with \( e_1 = (v_1, v_2) \) and \( e_2 = (v_2, v_3) \), \( e_1 \) is executed before \( e_2 \) if \( \rho(e_1) < \rho(e_2) \).

An example of a sequence diagram with cascaded loops and the related CDG is shown in Figure 3. The algorithm for the transformation of loops starts at the most inner loop, creating a backward edge between the last and the first fixed communication endpoint with the highest priority and an execution number of one less than in the sequence diagram. The edge pointing out of the loop receives a lower priority than the back-pointing one. This algorithm uses the information about the order hierarchy of the loops to process recursively the whole graph. In the case of non-fixed messages at the beginning or the end of the loop fragment, e.g. because of an optional fragment, the loop is unrolled.

**Figure 3. Transformation of loops**

For the expression of the inter-process association of branches in different processes, we introduce relations. Edges with the same relation are derived from the same fragment of a combined fragment as shown in Figure 5. So they are semantically coupled in UML and into the CDG. To associate relations, relation lists \( R_L \) are introduced. These are later used for the formal determination of the compatibility of different paths in the CDG.

- \( R_{CDG} \subseteq E_{CDG} \), with \( E_{CDG} = E_{CDG} \times E_{CDG} \) with \( n \geq 2 \) the set of relations between edges \( e \in E_{CDG} \).
- \( RL_{CDG} \subseteq R_{CDG} \), with \( R_{CDG} = R_{CDG} \times E_{CDG} \) with \( n \geq 2 \) the set of associated relations \( r \in R_{CDG} \).

The function \( r_{CDG}(e) : E_{CDG} \rightarrow R_{CDG} \) is defined by the algorithm presented in Figure 4. This prevents the detection of system properties like deadlocks that are semantically impossible in the specification. Nevertheless, our method is applicable for the recognition of deadlocks caused by the communication order, the intervals of the execution times and the blocking property of communication nodes. These deadlocks are often difficult to recognize using simulation based approaches. The combined fragment \( PAR \) is translated using relations and best-case and worst case combinations of the fragments.

**Figure 4. Compatibility of edges**

Later, we need a method for the determination of associated edges to attend the specification whilst the formal analysis and the simulation. Therefore, we use the following method to decide the match between edges using the information stored with relations. The compatibility of a set \( S_r \) of edges \( e \in E_{CDG} \) is defined by the algorithm presented in Figure 5.

**Figure 5. Transformation of alternative expression**

Afterwards, the timing information contained in the sequence diagram have to be used to annotate the edges in the CDG. These information can be interpreted as constraints for the specified case or they can be initially derived by existing implementations with static timing analysis. The different possible applicable expressions of time in sequence diagrams have to be converted to durations as it is needed in the CDG. Afterwards, we perform WCET/BCET analysis with respect to the
annotated values between communication in the sequence diagram. The flattening of method calls and inter object communication speeds up the simulation, besides. Precondition for our later analysis steps is the consideration of the different statements and the check if the annotations lead to an absolutely defined CDG.

The definition of multiple use cases using sequence diagram implies the transformation of them to a single CDG. They can be coupled by use case diagrams in the UML model.

UML state machines can contain hierarchical behavioral descriptions. An external switch between processes can be modeled to include the mutual exclusive usage of processing resources and operating system properties. These information can be included in the analysis model, but this is out of the scope of this paper.

4. Architectural Mapping

The architecture of the system defines location, type and interconnection of the instances of the system. The access on shared resources as communication busses can be analyzed with the knowledge about the processes, their communication schedule and their architectural mapping. Structured classes provide a flexible and powerful framework for the definition of system architectures. An essential feature for the description of communication structures is the ability to define ports and connectors between the instances of the system. The architecture provides properties of the platform model as communication latencies and durations for the communication analysis. Moreover, the information about shared communication media can be used as constraint for the conflict analysis. The results of the simulation based and formal analysis can evaluate different architectures.

Figure 6 shows two architectures for an SoC, mapping three communicating instances to structural elements. In both cases, they are connected to an on-chip bus, the difference is a dedicated communication line connecting P1 and P2 and the architectural refinement, using concrete platforms.

![Figure 6. Architectural modeling with UML2.0 structured classes/SysML assemblies](image)

The mapping between the functionality and modules of the architecture is expressed by deployment diagrams.

5. Communication Analysis

The approach presented in this section allows the determination of the temporal behavior of concurrent processes. The representation of the processes is done by communication and the timing behavior between the communication endpoints as they are considered in the CDG as nodes. The set of use cases defined at the design stage as sequence diagrams can be later used as a test suite for the evaluation of architectural designs in the stages of the architectural exploration and functional refinement. Objective of our communication analysis approach is the impact of blocking communication on the runtime behavior of processes. These blocking communication synchronize the involved processes. They are called synchronization point. Each communication edge embodies a potential candidate for a synchronization point. Obviously, the blocking communication partner has to be reached before the non-blocking one. Otherwise, this communication does not synchronize the participating processes. Based on the communication dependency graph, a synchronicity condition can be formulated in order to describe the influence of blocking communication on the inter-process synchronization. Goal of our analysis is the estimation of the time, the control flow has to wait in blocking nodes for the arrival of communication. These times are represented by slack values. These variables are used in an iterative process for the calculation of further slacks and in later analysis for the determination of performance parameters. For the efficient analysis of the slack values, we are creating a system of equations including slack variables to represent minimum and maximum wait times.

Considering edges with vary executing times, the following equations can be formulated to calculate the minimum $x_k$ and maximum $\tau$ slack of a communication $C_i$ with blocking node $v_b$ and non-blocking node $v_n$:

\[
\begin{align*}
x_k &= \min(path(S_pre(v_b, v_b) \rightarrow v_n)) \\
\tau &= \max(path(S_pre(v_b, v_b) \rightarrow v_n))
\end{align*}
\]

The set of nodes with minimal and maximal edge weights and slack values between $v_1$ and $v_2$ is denoted by $path(v_1 \rightarrow v_2)$. The set $S_pre(v_b, v_n)$ refers to all previous synchronization points from which the communication nodes $v_{nb}, v_{nb}$ can be reached directly without passing any other synchronization point. The communication analysis of the whole graph is an iterative process creating a result graph containing the slack values. The algorithm shown in Figure 7 describes our communication analysis approach.

```plaintext
A_0 := \{1, 2, ..., n\};
v_0 := \emptyset;
S := (A_0, C_0);
G := (A_0, C_0, V_0);
V.push((v_0, A_0, C_0));
while not A = update(A_pre);
if (A_pre, C_pre) \in S then
  for each c_e, e \in A_pre with (c_e, e) = e \in E_COM
    c := comana(e, A_cur, C_cur);
    c_cur := C_cur \cup c
    create edge k = (v_pre, v_cur)
    V.push((v_pre, A_cur, C_cur))
    S := S \cup (A_cur, C_cur);
    G := G \cup (A_cur, C_cur);
  else
    v_cur := G(A_cur, C_cur);
    create edge k = (v_pre, v_cur);
```

![Figure 7. Communication analysis](image)

The function update([v_1, ..., v_n]) returns a set of predecessor states of the state [v_1, ..., v_n]. The predecessor states must consider the compatibility of the relations of the edges as defined...
in (4). The function \( \text{comana}(e, [v_1, ..., v_n], [\{x_1\}, \{y_1\}], ..., [\{x_m\}, \{y_m\}]) \) calculates the value of the minimal and maximal slack values of communication \( e \) based on the state of the system \([v_1, ..., v_n]\) and the current slack values \(\{x_1\}, \{y_1\}], ..., \{x_m\}, \{y_m\}\) using (1) and (2).

6. Performance and Conflict Analysis

When we consider distributed real-time critical systems, the impact of a conflict on a shared communication medium can effect a violation of a hard deadline. As a result, a conflict free access to the communication medium has to be guaranteed. One possible next step in our analysis towards the assess of a given system specification is the determination of a communication schedule, that describes the order of the communication of the processes using the results of the communication analysis. This kind of information can be used to gain knowledge about conflicts on shared (communication) resources. The idea is, that the determined synchronization points set the parallel processes into relation and allow to define a temporal order of all communication. The position of communication in a schedule depends on their duration and the jitter of the execution times and slacks in the control flow in relation to any other communication. The result of the analysis can be presented by an access graph with potentially overlapping intervals. In the case of an overlap, it does not automatically imply the parallel access to a shared resource at the same time. Further analysis must attend the sequentiality of the communication ordered by the control flow in the CDG. Using the knowledge about conflicting accesses, the specified architecture of the system can be examined. Our approach for conflict analysis has been presented in [13]. Further investigations based on the communication analysis address the area of performance aspects. The influence of parallel processes to one process of the system is contained in the slack variables. To determine the worst case turn-around time of one process, only the path latencies of this process together with the slack variables have to be considered. Our approach has been presented in [14]. Moreover, guaranteed data rates the system can process can be determined by calculating the WCRT between the communication node receiving the communication from the environment. Further investigations towards the environment of a system can be performed by including environment models like event streams in our analysis approach by transforming them to our analysis format. So it is possible to evaluate a system specified by UML/SysML with common analysis models.

7. Simulation Model Generation

Simulation provides another tool to estimate the performance of a system model. Although simulation is accurate with completely specified models, it is not guaranteed to cover any corner case. Based on the simulation, monitoring of system properties as data throughput or resource usage can help to evaluate different designs. Because of the structured representation, the ability to consider aspects of hardware and software and to model at different abstraction levels, we use SystemC [12] as simulation language. Therefore we present a flexible and fast framework for the high level simulation based analysis of the performance of a system. The architectural model provides the structure of the SystemC model and the information about the mapping of processes to modules and of communication to shared or dedicated media. Furthermore, communication latencies are extracted from the architectural information. The internal control flow of the processes is adopted from the CDG. From the functional point of view, execution times and communication latencies are selected locally in the given intervals during simulation. The previously performed WCET/BDET analysis reduces the amount of information to be used. Globally, decisions about the execution number of loops concerning multiple processes have to be made. Branches are globally selected by using the relations annotated at the edges. So the data flow independence can be kept. For monitoring the behavior of the system, we insert traceable signals for the arrival and departure of the control flow at communication nodes. The events of sending and receiving delayed communication are traced as well. We allow asynchronous communication by buffering data to prevent data loss. The framework allows to monitor conflicts onto shared media.

8. Experimental Results

In this section we start with a small example presenting our flow for performance analysis of UML models. Figure 8 depicts a system consisting of two processes annotated with timing information. As an illustrative example, our objective is the determination of the maximum data rate the system can process. The first step performs the transformation of the UML model to a CDG that is depicted right of the sequence diagram.

For the description of a temporal environment we include an additional process in the formal system model that periodically sends a signal to node \( R_1 \) beginning after ten units of time. We start the analysis with the determination of the slack variables. Relating to equation (1) and (2) we calculate minimum and maximum slacks of the communication.

\[
\begin{align*}
\tau_1 &= l_{\text{min}}(\text{path}(I_e \rightarrow S_i)) - l_{\text{max}}(\text{path}(I_1 \rightarrow R_1)) \\
\tau_2 &= l_{\text{max}}(\text{path}(I_1 \rightarrow S_2)) - l_{\text{max}}(\text{path}(I_2 \rightarrow R_2)) \\
\tau_3 &= l_{\text{max}}(\text{path}(I_1 \rightarrow R_1)) + x_1 + l_{\text{min}}(\text{path}(R_1 \rightarrow S_2)) - l_{\text{max}}(\text{path}(I_2 \rightarrow R_2)) \\
\tau_4 &= 9 + 1 + 56 - 12 = 54 \\
\tau_5 &= 6 + 4 + 92 - 8 = 94 \\
x_3 &= \{142, 212\}
\end{align*}
\]

The maximum continuous data rate the system can process is defined by the worst case response time of process \( P_1 \).
We analyzed the runtime of the functional implementation between communication events on the relating processor models. We manually annotated timing information to the memories vmem_w, bmem_w, vmem_r, bmem_r and the stack vstack. The first result is that all components are able to handle the data rate of the previous component in the system. This information is prerequisite for the prevention of data loss caused by desynchronization of the components. The tool calculated, that the system is guaranteed capable to process a package every 188,092 µs. After using the same system without data caches, we calculated that the system is capable to process a package every 278,569 µs.

For both configurations our implementation according to Section 6 determines a static allocation of the communication to two busses that is sufficient for guaranteed conflict free access. Further modification of the architecture or the functional implementation can be made towards design space exploration. These examples show the applicability of our formal analysis approach and its value in the area of electronic system level design.

9. Conclusion

The complexity of distributed SoC is steadily increasing and demands for an early validation of constraints relating to a given architecture. We introduced an approach that is able to analyze the performance of systems modeled with UML/SysML even at high abstraction levels. We applied our methodology to transform UML models to our formal analysis model on time-annotated sequence diagrams and structured classes. Extensions to the analysis model were introduced to keep the semantics of the specification. We presented methods to analyze the performance of the transformed models with regard to the extensions. An other possibility for performance analysis is simulation. We described the creation of fast simulation models for monitoring system properties, performance parameters and constraint validation. The analysis approach enables the evaluation of functionalities and architectures. Our methodology allows design refinement and provides the link to design space exploration.

References
