Contracting Challenges for System Design and Integration

Mischa Möstl
IDA - Institute of Computer and Network Engineering
Technische Universität Braunschweig
moestl@ida.ing.tu-bs.de

Rolf Ernst
IDA - Institute of Computer and Network Engineering
Technische Universität Braunschweig
ernst@ida.ing.tu-bs.de

ABSTRACT

In this paper we highlight challenges of the applicability of contracting (based on assumptions and guarantees) for cyber-physical systems design. We illustrate in an example the limitations of an entirely composability-centered contracting approach. An alternative approach is subsequently proposed and applied to the presented example to illustrate that it is capable of handling the limitations demonstrated for the composable approach.

1. INTRODUCTION

Updatability and extending the functionality of devices in the field has become an established feature for consumer devices such as smartphones. Bringing this feature to complex cyber-physical systems (CPSs) such as cars, however, involves a number of challenges. Besides the obvious task of securely deploying an update, changes to a CPS often have safety implications. I.e. the whole CPS must be functionally safe after the update has been deployed. Either because the update added a safety critical functionality or since it was already present in the CPS before the update. Functional safety standards require for such applications that freedom of interference between critical applications and the rest of the system must exist [4, 7].

Similar to [4] we assume a V-model based development flow. Even in a conventional lab-based approach for such an update process several steps must be carried out to establish this. For in-field update abilities our research hypothesis is that applications are developed independently from the platform on the descending branch of the V, and the platform alone decides on the integration and deployment of updates. Assuming this, the system must automatically handle some design steps as part of the integration tasks on the ascending branch of the V, thus verifying a number of quite heterogeneous requirements. In order to be able to check requirements automatically they must be formally tractable.

A recent approach to formally specifying requirements for system compositions is contracting. The method assumes that for each component that is part of an update a contract is negotiated, such that all contracts for existing components meet their requirements as well as those of the update. In order to achieve this, contracts specify a set of assumptions, i.e. preconditions, such that a set of guarantees, i.e. postconditions, is provided given that the assumptions are fulfilled. The general understanding of this concept assumes that a contract is a bilateral agreement between two components on an interface and its specification, i.e. it suggests that contracts are negotiated between all pairs of components that share an interface. Further the concept favors composability of components, i.e. that any property of a component proven in isolation also holds in the integrated system as long as assumptions and guarantees, i.e. pre and postconditions, match at a service interface. Although this works very well for functional service interfaces, e.g. data types or value ranges, between components, it is a problem for non-functional aspects such as safety, timing, etc., which are typically global properties. One might argue, that this can be overcome by choosing the right abstraction layer with sufficient detail to model and tackle this effect. However, it remains that composability on higher layers cannot be achieved, especially for global system properties. Most prominent examples are systems with stateful shared resources, such as switched networks or caches which exhibit complex behaviors that are hard to abstract away and encapsulate in a component. Furthermore, this fact highlights the second issue of what the appropriate granularity for a component and its interfaces is. Section 2.1 illustrates this with an example.

The reason why choosing the "right" abstraction and the composability property are problematic are cross-layer effects, e.g. the dependence of execution times on the cache behavior. The reason therefore is the fact that the cache behavior is dependent on all the applications reading and writing on it, i.e. also applications that are not party to the respective (bilateral) contract. Cross-layer effects lead to dependencies which become effective when components are integrated and connected in a real system. These dependencies in the overall system then undermine the properties of a component, that where assumed static or verified when it was assessed in isolation, e.g. executing or simulating a component to estimate the execution time.

It can be argued that estimates on internal and component interface parameters can be tailored to a true worst-case design and contracts can be built accordingly it still requires composability for any global property. While full-scale worst-case design might be tolerable for high assurance system, e.g. in avionics, it is in general not applicable for a majority of CPS applications. Due to the overhead introduced by overprovisioning, it is in general deemed to expensive and thus unnecessary for all applications. With the only exception, if sufficient independence of implementation

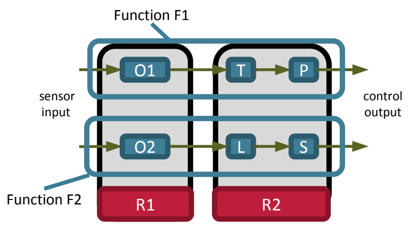

Figure 1: Two driver assistance functions integrated on a platform with two resources

between these and the critical functions can not be argued otherwise.

2. CONTRACTING FOR CPSs

One motivation for contracting is that responsibilities in a system scale development process can be clearly attributed to component suppliers by Original Equiptment Manufacturers (OEMs) or from tier-1 to tier-2 suppliers. The idea for a V-model based development process is that requirements are formulated on the descending branch and contract adherence is checked during integration on the ascending branch. In this manner non contract-conformant components and the responsible parties can easily be identified.

The other application of contracting is to support automation of integration. By enhancing e.g. software changes with a contract, a managing entity can then integrate the changed components based on the contracting information.

2.1 An Example

Contract-based composability is often favored for software modules. Composable interfaces seem particularly suitable for functional properties as for these all lower levels and esp. hardware effects are removed by abstraction. However, as already indicated, CPSs in general are not composable on higher layers due to the lack of composability for global properties, since these are undefined. Cross-layer effects that lead to dependencies once components are assembled in a system can undermine the properties that composable interface contracts try to provide.

Consider the example shown in Figure 1, which is a simplified case study taken from a fully electric research vehicle [5]. The solid blue boxes represent software components that are mapped to a resource platform (here) consisting of two resources R1 and R2. Multiple software components carry out two distributed functions F1 and F2, where F1 is a parking assistant (including parking spot detection), and F2 is a lane detection assistant. Software components O1 and O2 carry out object recognition and object masking (only O2) from camera and other sensor input on the resource R1. Resource R2 hosts the Trajectory calculation (T) and parking maneuvering (P), which forms the parking assistant function F1. Further R2 executes parts of the lane assist functionality, that consists of the two components lane detection (L) and steering parameter calculation (S) based on the object recognition and masking component (O2). For the sake of simplicity of this example, we consider the communication (green arrows), i.e. data-flow, event driven. As common for automotive systems, the resources are scheduled under fixed priority scheduling such as in AUTOSAR [1].

2.2 A Composable Approach

Assume we want to integrate the two functions contract based, i.e. provide a set of assumptions that if fulfilled give certain guarantees in return.

For software modules contract-based composability is often favored, since it allows building systems that are correct by construction. Composable interfaces seem to be particularly suitable for functional properties as for these all lower levels and esp. hardware are removed by abstraction. For instance, the T component will compute trajectories of a given accuracy and provide them in a fixed data format. The assumptions for this to hold are that at its input object maps of a given data format are available. For properties like this, contracts in assume and guarantee fashion can easily be (formally) formulated. Further, simple platform contracts can be formulated: For instance software components O1 and O2 are only available in a special binary format which is only supported by R1 and thus they must be mapped there.

Both examples formulate a purely functional property that can be checked bilaterally. I.e. this type of contracts works in a composable manner and the integration task would only have to check composability of contracts. However, this type of contracts neglects so called non-functional properties such as timing. Assume that the control algorithm implemented in T requires that new object data must not exceed a certain data age, i.e. delay from the point in time where the surrounding was sampled and processing the object map by T. Further assume that new data must arrive in a certain periodicity.

We see that the output timing is a function of the input function (event model) and the resource timing. I.e. it is neither a bilateral property between the software component O1 and T nor a property between T and its executing resource R2. This example already illustrates that CPS behavior, especially timing, is not composable on a higher layer of abstraction. In conclusion, contract mechanisms that require composability are not useful for practical CPS design, since effective CPS design on the one hand requires high-level overview as well as on the other hand guarantees from lower layers.

The example shows that especially cross-layer effects introduce dependencies between elements on different levels of abstraction, that make single-layer contract composability an intractable approach. To further highlight this with an example of a cross-layer effect: For instance, the thermal environment of a CPS can have a significant influence on the system behavior, e.g. the tolerable response time of a system can be violated if the CPU a component is executing on, is throttled at higher core temperatures. This is a primal example of the cross-layer nature of dependencies, since the thermal regime of a resource is typically not part of the timing model. In the example case it is rather the fact that the timing model is dependent on the Worst-Case Execution Time (WCET) estimation which in turn depends on other models such as the thermal regime or cache behavior.

2.3 An Alternative Approach

As we have seen a contracting mechanism that solely depends on composability of interface contracts cannot meet the demands of the integration tasks for CPSs. Therefore, the scope of contracting must be extended such that also global system properties can be verified for a component composition. Besides contracts that specify well defined in-

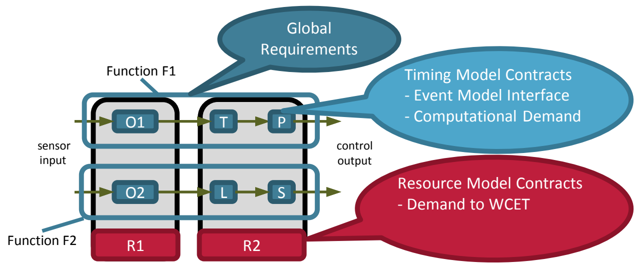

Figure 2: The example system from Figure 1 extended by model contracts. The figure shows them conceptually.

terfaces, contracts that specify models of a component are added to the contracting-based integration mechanism. As long as compatible models are chosen for different components, the models can be composed and further the composition can be analyzed. The model contracts thus enable to analyze global properties in a compositional analysis of the delivered models.

Furthermore, a specific model given by a contract can attach to other models that are provided by other contracts. I.e. if certain parameters of a model cannot be directly specified for one model, since the parameter is not an independent property of the component, it can be represented as a function which is evaluated in another model or an analysis thereof. In this way models can form a function-parameter hierarchy which can be compositionally evaluated, i.e. one model after the other until all global properties can be determined.

Applying this to our example from Figure 1, it allows us to specify a timing model for the two functions, i.e. for the individual components respectively. To demonstrate the idea, assume that each software component can be represent by an individual worker task. In conclusion each component has a contract that represents it as a task that has precedence constraints due to the dataflow on the components interfaces. The trailing tasks of each function chain, i.e. for O1 and O2, can be assigned event models as parameters that describe their activation period.

Yet in order to compute a Worst-Case Response Time (WCRT) on each chain, further information is necessary. Timing analysis methods such as Real-Time Calculus (RTC), Compositional Performance Analysis (CPA) or MAST therefore require a WCET parameter [8, 3, 2].

Since the WCET is not an independent property of a task but rather the combination of task and at least the resource it is executed on, we need an additional model that describes the relation of the demand a task has and the capabilities of the resource. A sufficient model interface for this can be provided by a model contract for the resources R1 and R2. This form of setting models in context allows also to address the previously introduced example considering thermal interference. Since now, not all information must be squeezed into a composable abstract interface contract, the WCET estimation model can query an appropriate model. The function parameter relation of the models then allows to evaluate the end-to-end WCRT compositionally by querying the appropriate WCET parameter for an analysis.

By also contracting the parameters of a model a high degree of flexibility is achieved. The benefits, besides the flexibility, are that the compositional approach can be applied at high levels of abstraction and is still formally tractable. The models supplied by the contracts on higher abstraction levels can be refined by the function-parameter hierarchy later on, where values can be fixed by contracts on the appropriate design levels.

Table 1: Model Parameters for the system in Fig. 1

	O1	O2	T	P	L	S
Event Model	Period=10, Jitter=1	Period=100 Jitter=5	propagated			
Priority[1]	1	2	1	2	3	4
WCET (normal)	5	29	1	1	10	16
WCET (throttled)	5	29	2	2	20	32

[1] lower value indicates higher priority

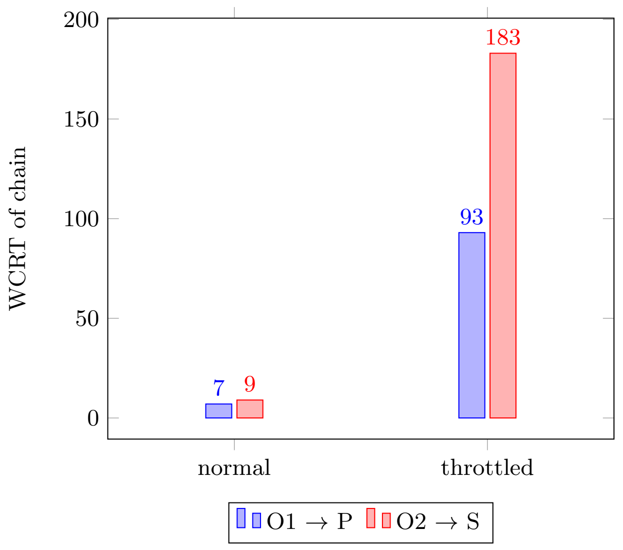

Figure 3: WCRTs of the function chains of the example system

We demonstrate this with the example system. Therefore, we select a set of parameters for the system that we provide in Table 1. Note that we assume that the WCET is not part of the timing-model contract each (software) component supplies, but an artifact of the combination of resource and timing-model contract. This way the resource model can supply different WCET values, depending on the thermal conditions of the resource.

To analyze the resulting construct we apply the analysis described in [6]. The results for the WCRT are displayed in Figure 3. As an integration check these would have to fulfill the requirements of a function contract, i.e. for the chain of software components.

3. CONCLUSION

In order to apply the outlined compositional contracting mechanism in a fully automated way, formalization of model transformations and model interfaces are necessary. I.e. how to automatically attach a WCET estimation model to a timing analysis and software component models. Further, if such interfaces also require transformations of the used models a proof of correctness for these transformations seems in order.

It has to be noted that the exemplified process adapts neatly to established company processes since it splits responsibilities instead of aggregating them in one component interface contract. In the compositional approach, information is only obtained when necessary for an analysis, i.e. it is obtained by function evaluation of a contract. Second, the information is obtained directly from the respective source, i.e. from the responsible model describing the property.

It is evident that for the integration task in complex CPSs more global properties than timing are of interest. We believe that the model-contract approach allows to evaluate the requirements of all the relevant viewpoints in appropriate models and thus is able to perform equivalent requirements checks as in a manual V-model process.

Acknowledgements

The authors would like to thank the other members of the DFG funded research unit *Controlling Concurrent Change (FOR 1800)* for their support.

4. REFERENCES

[1] AUTOSAR GbR. *Requirements on Operating System*, Release 4.2.1 edition, July 2015.

[2] M. G. Harbour, J. J. G. Garcia, J. C. P. Gutierrez, and J. M. D. Moyano. MAST: Modeling and analysis suite for real time applications. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 125–134, 2001.

[3] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis - the SymTA/S Approach. In *IEE Proceedings Computers and Digital Techniques*, 2005.

[4] International Organization for Standardization - ISO. *ISO 26262 - Road vehicles - Functional safety*, 2 edition, April 2011.

[5] A. Reschka, M. Nolte, T. Stolte, J. Schlatow, R. Ernst, and M. Maurer. Specifying a middleware for distributed embedded vehicle control systems. In *Vehicular Electronics and Safety (ICVES'14)*, Dec. 2014.

[6] J. Schlatow and R. Ernst. Response-Time Analysis for Task Chains in Communicating Threads. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–10, Apr. 2016.

[7] The International Electrotechnical Commission - IEC. *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2 edition, April 2010.

[8] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *The 2000 IEEE International Symposium on Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva*, volume 4, pages 101–104 vol.4, 2000.