Behavioural Composition Constructively Built Server Algorithms[*]

Pratyush Kumar and Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
E-mail: {pratyush.kumar, thiele}@tik.ee.ethz.ch

ABSTRACT
Composability and compositionality are well recognised as key enablers in rigorous design and analysis of complex systems. We argue that existing work on these enablers, specific to real-time systems, has exclusively focussed on design and analysis of *structural* composition, by which we refer to composition of either separate software tasks or compositions thereof on a shared platform such as a processor. Though structural composition is common and likely the most useful such composition, we make the case of an altogether different kind of composition which we refer to as *behavioural* composition. As a specific example of behavioural composition, we discuss how to constructively build complex server algorithms, called Demand Bound Servers (DBS), by composing constituent simpler server algorithms, even hierarchically. As a result of such composition, we can build server algorithms to more tightly match the requirements of tasks they need to serve, which indeed is not possible with the simpler components themselves. This is an example of how new behaviour is emergent out of the composition. We remain curious if there are other such examples of behavioural composition of interest to real-time systems.

1. INTRODUCTION
Composability and compositionality are well recognised as key enablers in the rigorous design and analysis of complex systems. First we present the definitions of these closely related words as we use them. By composability, we refer to the compatibility of components to be assembled together with other components to form more complex systems, using certain interface operations while preserving certain properties. On the other hand, by compositionality, we refer to the principle that the analysis of a system can be derived from the analysis of constituent components along with certain interface rules. Indeed, these two principles often co-exist: To be able to analyse systems compositionally, we often depend on decomposing it into composable components.

Let us transpose these principles on to the systems of our interest, namely real-time systems wherein we are primarily interested in temporal properties. There is a wealth of literature that studies composition in real-time systems. The strongest case for composition has been presented by Kopetz *et al* in their advocacy of time-triggered architectures [1]. Such an architecture is "decomposed into nearly autonomous clusters and nodes, and a fault-tolerant global time base of known precision is generated at every node" where "this global time is used to precisely specify the interfaces among the nodes" [1]. In [2], the authors apply principles of time-triggered architectures to present a comprehensive design-flow for multi-processor system-on-chips.

The other major thrust to composable real-time systems comes from the design of resource isolation algorithms, otherwise also called servers [3], [4], [5]. The primary aim in using servers is to compose multiple tasks (or task-sets) while preserving temporal properties within each task, within established checks on schedulability. The servers thus act as wrappers to tasks (or task-sets) to enable composability and provide the interface rules necessary for analysis of the composed system. In a typical use case, a new server may be "safely" composed to an existing system if it satisfies certain interface conditions of schedulability alone.

The third important direction of focus has been in the compositional analysis of complex real-time systems. Such methods decompose a system into analysable components which are modelled in abstractions with suitable interface operations to combine (often hierarchically) such analyses. Several such abstractions have been proposed including timed automata [6], periodic resource model [7], dataflow models [2], arrival and service curves [8], period with jitter model [9] among others.

Let us establish a common thread in all the above works. A key commonality is the composition of *separate* tasks or resources. For instance, we compose two separate tasks onto a time-triggered architecture, or two separate servers on to the same processor, or two periodic resource models representing separate tasks into one periodic resource model. Indeed the tasks or resources composed can be related or may share a common operational objective, such as the software units of a car may together accomplish auto-pilot mode. But such a property is not an essential requirement of the composition, neither a consequence thereof. In this sense, the composition is done to build and analyse *bigger* systems. Differently stated, the key emergent property of such composition is a more developed structure. Thus, we refer to it as *structural* composition.

The need for such structural composition in real-time systems

[*]Copyright is held by the authors.

is apparent. There is the valid case of designing and optimising different applications in isolation and composing them later with minimal information exchange while guaranteeing validity of certification. Secondly, the sheer complexity of large distributed real-time systems necessitates the presence of clearly defined composition principles.

The ubiquity and utility of structural composition must not detract us from exploring other types of composition, if any. Indeed the nomenclature of structural composition was motivated by the presence of another kind of composition, which we term *behavioural* composition. *By this we refer to a composition principle wherein the key emergent property is a more developed behaviour, which was not realisable without the composition.* While structural composition is about a bigger system, behavioural composition is about a "better" system.

In this work, we concretise the concept of behavioural composition with a specific example, namely the design of Demand Bound Server (DBS) presented in [10]. Starting from an elementary class of DBS, it is shown that by proposed composition operations, the class of DBS that can be implemented is generalised. Such a generalisation is the emergent property of the composition. The utility of this emergent property is the ability to tightly serve task-sets within serves without compromising on the schedulability.

While we provide one specific example of behavioural composition, we remain curious if there are others. For instance, can we compose two scheduling policies to generate a new scheduling policy with different, otherwise unrealisable, properties. Or can we compose two different speed scaling laws to manage energy and temperature into one speed scaling law. To reiterate, in all of these potential ideas, we do not aim for structural multiplicity rather we aim at behavioural speciality. For instance, the speed scaling law that is actuated on the processor is still one law, but it may be composed of two differently designed and appropriately interfaced speed scaling laws.

The rest of the paper is organised as follows. We discuss the definition of Demand Bound Server and present the composition operations in the Section 2. We present the emergent behaviour from this composition technique in the Section 3. Finally, we compare our behavioural and structural compositions in Section 4.

2. DEMAND BOUND SERVERS

In this section, we will describe the definition of Demand Bound Server (DBS), discuss the implementation of a specific kind of DBS, namely Shifted Periodic Demand Bound Server (SP-DBS), and the operations on DBS.

2.1 EDF-Servers

We refer to servers which are designed to work with EDF as the underlying scheduling policy as EDF-servers. Examples of existing EDF-servers are Constant Bandwidth Server (CBS) [11], Total Bandwidth Server (TBS) [3], Dynamic Sporadic Server [12], among others.

Typically, the interfacing between the server and the underlying scheduler can be generalised as follows. The server specifies at all time instants, up to three quantities (a) a deadline d, (b) a budget c, and (c) a re-insertion time r. The deadline is used to compete for the resource under the EDF policy. The budget is the maximum amount of execution the server can receive before it is inactive, i.e., not contending for the resource. The re-insertion time is the next time instance when the server will contend for the resource, if it is currently non-contending. At times when the server becomes inactive, either due to depleted budget or an empty task queue, the scheduler sends the server a value δ which is the amount of execution it has received since the last such inactivation. Thus, the tuple (c, d, r, δ) represents the possible two-way interactions between an EDF-server and the scheduler. The role of the server algorithm is to appropriately modify the interface variables c, d and r, by utilising server parameters and the values of δ available from the scheduler.

2.2 Demand Bound Server: A Definition

The Demand Bound Server (DBS) was designed to generalise the existing EDF-server [10]. Central to the definition of the server is the concept of demand bound function [13] which is defined as follows.

DEFINITION 1 (DEMAND BOUND FUNCTION). *A task has a demand bound function,* $\mathtt{dbf}$, *if for any* $\Delta > 0$, *in an interval of time* $[t, t+\Delta]$ *for any* $t > 0$, *the sum of the execution times of all jobs that arrive not earlier than* t *and have deadline not later that* $t+\Delta$, *does not exceed* $\mathtt{dbf}(\Delta)$.

A Demand Bound Server (DBS) is characterised only by a demand bound function denoted as $\mathtt{dbf}_s$ (the 's' subscript denotes server parameters). The definition of the server is then given by the following two properties based on this demand bound function. (For a more formal definition of these properties, please refer to [10].)

DEFINITION 2 (DBS PROPERTY I). *A DBS characterized by the demand bound function* $\mathtt{dbf}_s$ *must not request execution on the resource more than that of a task with demand bound function* $\mathtt{dbf}_s$.

DEFINITION 3 (DBS PROPERTY II). *When a task (or a task-set) with a (cumulative) demand bound function not larger than* $\mathtt{dbf}_s$ *is served by a DBS with demand bound function* $\mathtt{dbf}_s$, *then all jobs must meet their respective deadlines.*

As a consequence of these properties, we can (a) verify schedulability of a DBS on a resource, and (b) compute response times of jobs served by the DBS [10].

Any EDF-server which satisfies the above two properties is deemed a DBS with the characteristic demand bound function. For instance, it is possible to show that the commonly used servers such as CBS and TBS are indeed DBS. In this sense, it is a generic definition of a large class of EDF-servers. The key result of defining such a generic class is the following result on optimality [10].

THEOREM 1 (TIGHTNESS OF DBS). *If a set of tasks, each characterized by a demand bound function is EDF schedulable, then if we serve each task by a DBS characterised by the demand bound function of that task, the servers are schedulable and real-time requirements of all tasks are met.*

The above result implies that it is always possible to identify a configuration of DBS such that they can be used to serve a set of tasks while not affecting schedulability even under the optimal EDF policy. This implies that the cost of isolating the tasks with servers does not affect schedulability. However, this precludes that we are indeed able to *implement* server algorithms for any given DBS. This is the topic for the rest of the section.

2.3 SP-DBS

Though DBS represents a wide class of EDF-servers, a server algorithm which implements a DBS with a generic $\texttt{dbf}_s$ can have a prohibitively high implementation cost. Considering that the server algorithm is an additional overhead on the scheduling, the choice of algorithm is sensitive to the implementation cost. We will now discuss a Shifted-Periodic Demand Bound Server (SP-DBS) which is a specific kind of DBS that has been shown to have a small implementation overhead [10].

SP-DBS being a DBS has a characteristic demand bound function, which in this case is parameterised by the tuple (P, Q, D). In terms of these parameters the demand bound function is given as

$$\texttt{dbf}_s(\Delta) = \max\left\{0, \left(\left\lfloor \frac{\Delta - D}{P} \right\rfloor + 1\right) \times Q\right\}. \qquad (1)$$

In words, the above function is periodic with period P, increasing by Q every period, but with an initial offset of D. Otherwise stated, it is the demand bound function of an explicit deadline task with period P, execution time Q and deadline D. Note that a SP-DBS generalises the utilisation-based server algorithms such as the Constant Bandwidth Server (CBS). Indeed the server algorithm of SP-DBS is markedly different from that of say CBS.

The server algorithm for a SP-DBS is presented in [10]. It has been shown that the overhead of implementing the server algorithm in terms of number of instructions and memory space required are both reasonable. Hence, SP-DBS can be qualified as a DBS with a realisable server algorithm. However, we will like to further broaden the class to include a larger set of servers. This is the aim in the rest of the section.

2.4 Operations on DBS

We will now discuss operations that can be performed on one or more DBS, which will later be used to demonstrate behavioural composition in DBS. These operations are realised by *adapters* which suitably modify the interface variables, namely the tuple (c, d, r, δ).

2.4.1 Left-Shift Operation

The left-shift operation modifies a DBS to have a larger demand bound function obtained by shifting it to the left by a parameter τ. The left-shift operation is realised by modifying the interface variables as described below.

DEFINITION 4 (LEFT-SHIFT). *A server S is said to implement* left-shift by τ *of a DBS S_1, denoted as $S := (\overset{\tau}{\leftarrow} S_1)$, if the adapter of S implements the following interface*

$$\begin{aligned} d &:= d_1 - \tau, \\ c &:= c_1, \\ r &:= \max(t, r_1 - \tau), \\ \delta_1 &:= \delta, \end{aligned} \qquad (2)$$

where t denotes the current time.

For the left-shift operation it can be shown that the resultant server is indeed a DBS with the shifted demand bound function, as long as the resultant server is schedulable.

THEOREM 2. *Let S_1 be a DBS with demand bound function $(\texttt{dbf}_s)_1$, such that $(\texttt{dbf}_s)_1(t+\tau) \le t, \forall t \ge 0$. Then, $S := (\overset{\tau}{\leftarrow} S_1)$ defines a DBS with a demand bound function $\texttt{dbf}_s$ given as*

$$\texttt{dbf}_s(t) = (\texttt{dbf}_s)_1(t+\tau), \quad t \ge 0. \qquad (3)$$

2.4.2 Min Operation

The min-operation aggregates a set of DBS into one DBS with the demand bound function equal to the minimum of the demand bound functions of all the servers. The min operation is realised by the following changes to the interface variables.

DEFINITION 5 (MIN-COMPOSITION). *A set of DBS $\{S_1, S_2, \ldots S_n\}$ is said to be min-composed to form a server S, denoted as $S = S_1 \wedge S_2 \wedge \ldots \wedge S_n$, if the adapter of server S implements the following interface*

$$\begin{aligned} d &:= \max(d_1, d_2, \ldots d_n), \\ c &:= \min(c_1, c_2, \ldots, c_n), \\ r &:= \max(r_1, r_2, \ldots, r_n), \\ \delta_i &:= \delta, \qquad \forall i \in \{1, 2, \ldots, n\}. \end{aligned} \qquad (4)$$

As before, it can be shown that the aggregate server is indeed a DBS with the desired demand bound function.

THEOREM 3. *Let $S := S_1 \wedge S_2 \wedge \ldots \wedge S_n$, then the server S is a DBS characterized by demand bound function $\texttt{dbf}_s$ which is related to $(\texttt{dbf}_s)_i$, the demand bound function that characterizes server S_i, $i \in \{1, 2, \ldots, n\}$, as follows*

$$\texttt{dbf}_s := \min\left((\texttt{dbf}_s)_1, (\texttt{dbf}_s)_2, \ldots, (\texttt{dbf}_s)_n\right). \qquad (5)$$

Note that in both cases, the adapters required to perform the operations on the interface variables are indeed simplistic, in terms of the overhead they additionally admit. Thus, performing these inexpensive operations will retain the releasability of the underlying DBS, for instance SP-DBS.

3. EMERGENT BEHAVIOUR FROM DBS COMPOSITION

In this section, we discuss how complex server algorithms can be constructively built by composing DBS. We will first discuss two examples of specific problems and highlight the new emergent behaviour and finally formalise the class of all server algorithms that can be built using such composition.

3.1 Serving Tasks with Jitter

Consider a task-set of two periodic tasks T_1 and T_2. Both tasks have a period and relative deadline of 10. The execution times of the two tasks are 5 and 4, respectively. Task T_1 in addition can have a jitter of up to 5. It can be verified that the two tasks can be scheduled under EDF [14]. However, using utilisation-based

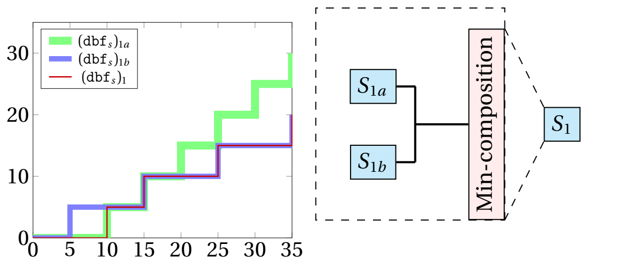

Figure 1: Serving tasks with jitter using operations on SP-DBS

servers, such as Constant Bandwidth Server (CBS), the two tasks are not schedulable, as the peak utilisation of the tasks (0.67 and 0.4, respectively) add up to more than 1. Neither are these tasks schedulable with SP-DBS, as both are implicit deadline tasks with deadline equalling the period.

We will now use composition to constructively build the server algorithms required to serve the considered tasks. Let task T_2 be served by a SP-DBS denoted S_2 with parameters $(P,Q,D) = (10,4,10)$. It is clear to see that this suffices to meet the deadlines of the task T_2. To serve task T_1 we construct the server S_1 built by two servers S_{1a} and S_{1b}. Server S_{1a} is a SP-DBS with parameters $(P,Q,D) = (10,5,5)$ and server S_{1b} is a SP-DBS with parameters $(P,Q,D) = (5,5,10)$. To build S_1 we min-compose the two SP-DBS, namely S_{1a} and S_{1b}. The demand bound functions of these servers are shown in Figure 1. As can be seen from the figure, the $\mathtt{dbf}_s$ of S_1 exactly equals the demand bound function of task T_1. Consequently S_1 can meet the deadlines of T_1, and in addition the two servers S_1 and S_2 are schedulable together (sum of demand bound functions is below the constant line of slope 1 through the origin).

By the process of composing the servers, we were able to build a server algorithm that realises an exact match of the demand bound function requested by a task and thereby leads to a schedulable system. Thus, composition generates a behaviour hitherto not realisable by existing servers. To reiterate, the servers S_{1a} and S_{1b} do not serve separate tasks, they work synergistically to realise one server that behaves differently.

3.2 Serving Tasks with Burst

Now consider another task set of two tasks T_3 and T_4. The two tasks have a period and relative deadline of 10. The execution times of the tasks are 5 and 2.5, respectively. Task T_3, in addition, can have a jitter of up to 5, exhibited by up to 4 consecutive jobs. It can be verified that the two tasks can be scheduled under EDF. However, using the standard servers such as Constant Bandwidth Server (CBS) the two tasks are not schedulable, as the peak utilisation of the tasks (0.8 and 0.25, respectively) add up to more than 1. Neither are these tasks schedulable with SP-DBS, as both are implicit deadline tasks with deadline equalling the period. It can be verified that min-composing SP-DBS also does not generate a schedulable system.

This is an example where we demonstrate the utility of operations on DBS, performed hierarchically. Let task T_4 be served by a SP-DBS denoted S_4 with parameters $(P,Q,D) = (10,2.5,10)$. It is clear to see that this suffices to meet the deadlines of the task T_4. To serve task T_3 we construct the server S_3 built by two servers S_{3a} and S_{3b}. The server S_{3a} is a SP-DBS with parameters

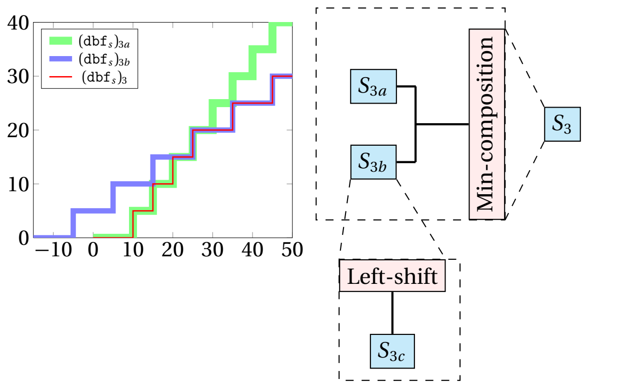

Figure 2: Serving tasks with bursts using operations on SP-DBS

$(P,Q,D) = (5,5,10)$. The server S_{3b} is the left-shift of a server S_{3c} by the parameter $\tau = 15$. In turn, the server S_{3c} is a SP-DBS with parameters $(P,Q,D) = (10,5,10)$. The demand bound functions of these servers are shown in Figure 2. As can be seen from the figure the $\mathtt{dbf}_s$ of S_3 exactly matches the demand bound function of task T_3. Consequently, S_3 can meet the deadlines of T_3, and in addition the two servers S_3 and S_4 are schedulable together (sum of demand bound functions is below the constant line of slope 1 through the origin).

Again, this example illustrates how we constructively built the server algorithm by hierarchical operations on multiple SP-DBS. Again, the emergent property of the composition is a more finely controlled behaviour.

3.3 The Class of Server Algorithms

We will now formally characterise the class of server algorithms that can be built by hierarchically min-composing SP-DBS or left-shifted SP-DBS. Since, the left-shift and min operations preserve the DBS properties, we can equivalently characterise the class of server algorithms by the class of the demand bound functions which characterise the respective server algorithms.

Firstly, note that the need to support more complex demand bound functions arises due to potential variability in the jobs' arrival pattern, such as jittery and bursty arrivals (as in pervious two examples). It is thus instructive to characterise such variable patterns. To this end, we will first characterise the concept of arrival curve from Network Calculus [15].

DEFINITION 6 (ARRIVAL CURVE). *The arrival curve α of a task is the smallest such curve where the number of jobs that can arrive in any interval of length Δ is no more than $\alpha(\Delta)$.*

Variable job arrival patterns of different types such as periodic jobs, with and without jitters, with and without bursts, can be compactly represented by arrival curves. A class of practically relevant such curves is given by the curve of a cascade of *leaky-buckets*. A leaky-bucket is parameterised with a rate r and fill capacity b. The curve of the leaky bucket is given by $\sigma(u) = \lceil ru + b \rceil$. It models the arrival pattern of a task with its period $1/r$ and a maximum number of jobs arriving in a burst of b. Leaky-buckets can be cascaded in series to obtain more detailed arrival patterns. The curve of two leaky buckets with curves σ_1 and σ_2 cascaded in series is given by $\sigma = \min(\sigma_1, \sigma_2)$. The physical interpretation of such a cascading is that if an input stream of jobs is passed through components serially, where each component restricts the rate and maximum burst of the stream of jobs, then the output stream has an arrival curve at most equal to the curve of the cascaded leaky buckets of respective parameters.

In most practical scenarios, arrival curves can be considered to be represented by the curve of a cascade of leaky buckets. If such a stream of jobs has a fixed (across jobs) worst-case execution time and a fixed (across jobs) relative deadline, then we show that the resultant demand bound function can be tightly matched by the demand bound function of a DBS constructively built in the proposed manner.

THEOREM 4. *The class of demand bound functions of servers that can be constructively built using SP-DBS is the class of demand bound functions of any task with an arrival curve as a cascaded leaky-bucket curve, and fixed worst-case execution time and relative deadline.*

PROOF. Given that the worst-case execution time and relative deadline of all jobs are fixed, the demand bound function of the task is only a scaled and right-shifted version of its arrival curve. This curve in turn is the minimum of a set of leaky-bucket curves. Clearly such a demand bound function can be obtained by min-composing multiple DBS, where each DBS models a single leaky-bucket. A SP-DBS, or a left-shift of one, can be chosen to have as demand bound function the curve of each leaky-bucket. □

Note that we have discussed the exact matching of the server and task demand bound functions. Such a match implies that the tasks can be isolated without any penalty in terms of schedulability, i.e., Theorem 1 applies. Indeed, it is possible to use a DBS to serve a task or a task-set with a cumulative demand bound function *smaller* than that of the demand bound function of the server. In such a scenario, we would pay for the isolation with a certain loss of schedulability, which one may well call a *schedulability gap*. Indeed, the endeavour of defining new server algorithms has been to reduce this gap.

4. STRUCTURAL AND BEHAVIOURAL COMPOSITIONS

Having described the constructive building of DBS, we will now present an example system to illustrate the differences between the two proposed kinds of compositions.

Consider a resource which uses timed-triggered architecture to serve two applications, or equivalently task-sets, which are to be independently developed. This is an example of structural composition where the main goal is *modularity in the design process*.

We only consider the second application A_2. This application runs on an EDF scheduling policy and supports a DBS S_7 and a task T_{56}. This is another example of structural composition where the main goal is *isolation during composition* as the jobs of task T_{56} will not be affected by overruns of the jobs served by S_7.

The task T_{56} in turn is the composition of two tasks T_5 and T_6 under the round-robin queueing discipline. This is another example of structural composition, where the main goal is *aggregation*

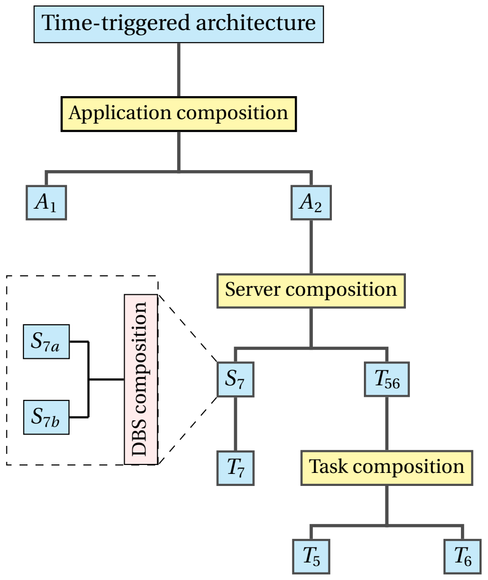

Figure 3: Structural and behavioural composition

of two streams of jobs into one under the interface operation of round-robin ordering.

Note how the three structural compositions enable separate software units to be supported on a common platform, while ensuring certain properties during the composition. In contrast, now consider the server S_7 serves a single task T_7 which has an arrival curve given as the curve of a cascade of two leaky-buckets. To ensure optimality under schedulability (Theorem 1), the $\texttt{dbf}_s$ of S_7 must match the demand bound function of T_7. To this end, S_7 is built using two appropriately parameterised SP-DBS under min-composition denoted as S_{7a} and S_{7b}. In this example the composition synthesis a new behaviour.

Such a construction of S_7 is possible because of two properties: (a) composability: the interface of min-composition enables to synergistically couple multiple SP-DBS (Definition 5), and (b) compositionality: the analysis of S_7 (in terms of its $\texttt{dbf}_s$) is modularly derived from the analysis of S_{7a} and S_{7b} (Theorem 3).

5. CONCLUSIONS

We presented the case for considering behavioural composition where the emergent property is behavioural speciality rather than structural multiplicity. As a specific example, we highlighted the constructive building of server algorithms with operations on Demand Bound Servers. The resultant class of servers was characterised, and was shown to greatly enrich the class of server algorithms which can be implemented while maintaining optimality of schedulability. The open problem is to validate the utility of defining behavioural composition by identifying other relevant examples of it.

Acknowledgements

This work was supported by EU FP7 projects EURETILE and PRO3D, under grant numbers 247846 and 249776, respectively.

6. REFERENCES

[1] H. Kopetz and G. Bauer, “The time-triggered architecture,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[2] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, “CoMPSoC: A template for composable and predictable multi-processor system on chips,” *ACM Trans. Design Autom. Electr. Syst.*, vol. 14, no. 1, 2009.

[3] M. Spuri and G. C. Buttazzo, “Efficient Aperiodic Service Under Earliest Deadline Scheduling,” in *IEEE Real-Time Systems Symposium*, IEEE Computer Society, 1994.

[4] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, “Resource kernels: A resource-centric approach to real-time and multimedia systems,” in *In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, 1998.

[5] G. Lipari and S. K. Baruah, “Efficient scheduling of real-time multi-task applications in dynamic systems,” in *IEEE Real Time Technology and Applications Symposium*, 2000.

[6] R. Alur and D. L. Dill, “A theory of timed automata,” *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[7] I. Shin and I. Lee, “Periodic Resource Model for Compositional Real-Time Guarantees,” in *RTSS*, pp. 2–13, IEEE Computer Society, 2003.

[8] L. Thiele, S. Chakraborty, and M. Naedele, “Real-time calculus for scheduling hard real-time systems,” in *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 101 –104, 2000.

[9] A. Hamann, M. Jersak, K. Richter, and R. Ernst, “Design Space Exploration and System Optimization with SymTA/S-Symbolic Timing Analysis for Systems,” in *RTSS*, pp. 469–478, IEEE Computer Society, 2004.

[10] P. Kumar, J.-J. Chen, and L. Thiele, “Demand bound server: generalized resource reservation for hard real-time systems,” in *EMSOFT*, pp. 233–242, ACM, 2011.

[11] L. Abeni and G. C. Buttazzo, “Integrating Multimedia Applications in Hard Real-Time Systems,” in *IEEE Real-Time Systems Symposium*, 1998.

[12] T. M. Ghazalie and T. P. Baker, “Aperiodic servers in a deadline scheduling environment,” *Real-Time Systems*, vol. 9, no. 1, 1995.

[13] S. K. Baruah, A. K. Mok, and L. E. Rosier, “Preemptively scheduling hard-real-time sporadic tasks on one processor,” in *IEEE Real-Time Systems Symposium*, 1990.

[14] F. Zhang and A. Burns, “Analysis of Hierarchical EDF Pre-emptive Scheduling,” in *RTSS*, pp. 423–434, IEEE Computer Society, 2007.

[15] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, vol. 2050 of *Lecture Notes in Computer Science*. Springer, 2001.