iDSL: Automated Performance Prediction and Analysis of Medical Imaging Systems*

Freek van den Berg, Anne Remke, and Boudewijn R. Haverkort

University of Twente, PO Box 217, 7500 AE, Enschede, The Netherlands
{f.g.b.vandenberg, a.k.i.remke, b.r.h.m.haverkort}@utwente.nl

Abstract. iDSL is a language and toolbox for performance prediction of Medical Imaging Systems; It enables system designers to automatically evaluate the performance of their designs, using advanced means of model checking and simulation techniques under the hood, and presents results graphically. In this paper, we present a performance evaluation approach based on iDSL that (i) relies on few measurements; (ii) evaluates many different design alternatives (so-called "designs"); (iii) provides understandable metrics; and (iv) is applicable to real complex systems. Next to that, iDSL supports advanced methods for model calibration as well as ways to aggregate performance results. An extensive case study on interventional X-ray systems shows that iDSL can be used to study the impact of different hardware platforms and concurrency choices on the overall system performance. Model validation conveys that the predicted results closely reflect reality.

1 Introduction

Embedded systems have faced a significant increase in complexity over time and are confronted with stringent costs constraints. They are frequently used to perform safety critical tasks, as with Medical Imaging Systems (MIS). Their safety is significantly determined by their performance. As an example of an important class of MIS, we consider interventional X-ray (iXR) systems, as built and designed by Philips Healthcare.

These systems create images continuously based on X-ray beams, which are observed by a surgeon operating a patient. Images need to be shown quickly for hand-eye coordination [11], viz., the surgeon perceives images to be real-time.

In earlier work, when the ASD method [7] was considered to be used for the design of iXR machines, we have evaluated their performance using simulation models, derived from the design specification by hand.

This paper presents a fully formalised performance evaluation trajectory in which we go from real measurements, via a formal model, to performance predictions, for many different designs, in a fully automated way. Starting point

* This research was supported as part of the Dutch national program COMMIT, and carried out as part of the **Allegio** project under the responsibility of the ESI group of TNO, with Philips Medical Systems B.V. as the carrying industrial partner.

for this evaluation are models expressed in the IDSL formalism, which has been introduced in [13] and is extended here to fit our new approach. From such models input for the Modest toolset [8] is automatically generated and results are visualized using Graphviz and Gnuplot. This is not only very efficient, it also brings advanced formal performance evaluation techniques, e.g., based on model checking of timed automata and Markov chains, and discrete-event simulation, at the fingertips of system designers, without bothering them with the technical details of these. Furthermore, the approach allows to efficiently predict the performance of a large number of design variants, to compare them, and select the best design given a set of constraints and measures of interest. However, note that in contrast to Design Space Exploration (DSE) [2], in which a few optimal designs are being searched for, we evaluated a large and fixed amount of designs.

Even though the presented approach is fairly general, we illustrate its feasibility on so-called *biplane* iXR systems, which comprise two imaging chains, positioned in perpendicular planes to enable 3D-imaging. They are currently implemented using two separate hardware platforms. However, for various reasons, e.g., costs, physical space, energy consumption and failure rate, it is worth investigating running the software for both image chains on shared (but more powerful) hardware. Hence, we use the above mentioned approach to predict the performance for shared hardware as a case study. Sharing hardware gives potential to concurrency, which may result in increased latency and jitter of images, which, in their turn, affect (perceived) system safety.

We have identified four key objectives that such an integral and fully automated performance evaluation approach should meet, i.e., it should

O1: use as few costly measurements as possible;
O2: be able to evaluate a large number of complex designs;
O3: present its predictions intuitively via understandable (aggregated) metrics;
O4: be applicable to real complex systems.

These objectives are realized through the following four contributions made in this paper. First, the model is calibrated using measurements and measurement predictions to rely on few costly measurements. In contrast, current Design Space Exploration approaches typically require many measurements to be readily available [10, 2]. Second, we use iDSL [13], a language and toolbox for automated performance evaluation of service systems, and extend it to support the prediction of unseen empirical cumulative distribution functions (eCDFs). The automation allows us to evaluate many designs, using Modest [8] for simulations, in line with previous work [9, 14]. Third, we use a variety of aggregation functions to evaluate designs on different aspects. Fourth, we conduct a case study on a real-life MIS, viz., Image Processing of iXR systems. We validate our model by comparing its predictions with corresponding measurements. Also, the predictions are used to gain insight in the performance of biplane iXR systems with shared hardware.

Paper outline: This paper is organised as follows: Section 2 provides the methodology of our approach. Section 3 describes how measurements are taken, predicted and applied. Section 4 sketches the iDSL tool chain and model. Section 5 presents the results of the case study. Section 6 concludes the paper.

2 Methodology

We specify our approach as a solution chain as depicted in Figure 1, consisting of three consecutive stages, viz., pre-processing, processing and post-processing. The iDSL toolbox automates these steps and connects them seamlessly.

During **pre-processing**, measurements are performed and execution times derived from them. They are performed for different iXR system configurations and yield large sets of so-called activities for every single design. An activity specifies, for a particular resource and a performed function, the time interval of execution. Activities are visualized automatically in Gantt charts [15]. They are grouped to obtain total execution times per function and in turn aggregated into so-called empirical cumulative distribution functions.

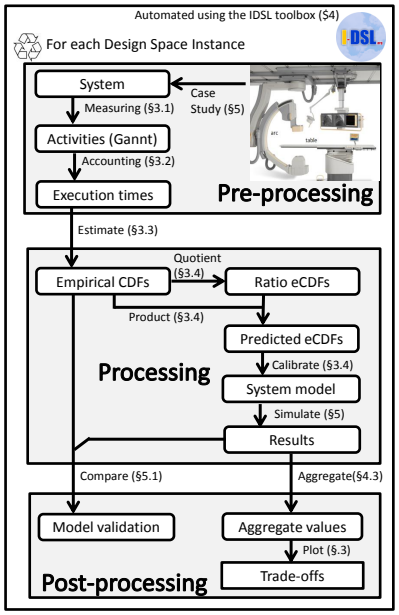

Fig. 1. The solution chain of the approach comprising pre-processing (performing measurements and deriving execution times), processing (predicting eCDFs and simulating) and post-processing (aggregate functions).

During **processing**, we start with many inverse eCDFs, all based on measurements, that cover all possible designs of interest. Many are used for model validation (explained below) and a fraction of them is used to predict new, inverse eCDFs. Hence, one may reason about the performance of many designs, while relying on only few measurements, in line with Objective **O1**.

Next, the iDSL model is executed to obtain performance results, in two steps: (i) iDSL predicts eCDFs for all designs and calibrates the model based on these eCDFs; and (ii) iDSL performs many simulations via the Modest toolset (see Section 4), yielding results for all designs, meeting Objective **O2**.

During **post-processing**, results are processed into aggregated, understandable metrics, facilitating the interpretation of the results (Objective **O3**).

3 Measurements & Emperical CDFs

In this section, measurements performed on design instances are used to predict the performance of other design instances, in four steps: (i) we perform measure-

ments that yield activities; (ii) these activities are grouped into execution times; (iii) these execution times are used to estimate emperical CDFs (eCDFs); and (iv) we predict eCDFs for the complete design space, relying on few estimated eCDFs. We discuss these 4 steps below in more detail.

3.1 Measuring activities on a real system

Measurements on embedded systems are typically performed by executing real program code augmented with stopwatches, during a so-called execution run. Stopwatches administer the starting and ending times of functions that run on different resources. We consider iXR systems that loop in cycles and perform a sequence of n image processing operations $(f_1, f_2, ..., f_n)$ on m parallel resources $(r_1, r_2, ..., r_m)$. Measurements lead to activities $Act : Res \times Cycle \times Time \times Time \times Func$ that specify a resource that performs a given function, in a certain cycle, during a time interval. Figure 2 visualizes this in a Gannt-chart.

The system designer requires iXR systems to meet two properties: (i) resources process only one operation for one image at a time, which reduces complexity but comes at the price of a reduced utilization; and (ii) iXR systems adhere to a strict FIFO scheduling policy to preserve the image order. Combined, these two properties ensure non-overlapping functions.

3.2 Grouping activities into execution times

To reduce complexity, we combine activities that perform the same functionality, in the same cycle, but on different resources, into one execution time; formally:

$$E_f(c) = \max\{t_2 \mid (r_i, c, t_1, t_2, f) \in Act\} \; - \; \min\{t_3 \mid (r_j, c, t_3, t_4, f) \in Act\}, \quad (1)$$

where f is a function, c the cycle, r_i and r_j resources, and t_1, t_2, t_3 and t_4 times. Execution time $E_f(c)$ may include time during which all resources idled. This may result from executing code without stopwatches, or a resource waiting for another resource. Either way, this idle time is attributed to $E_f(c)$ to not underestimate execution times. Finally, $E_\Omega(c)$ represents the overall execution time; formally:

$$E_\Omega(c) = \max\{t_2 \mid (r_i, c, t_1, t_2, f_i) \in Act\} - \min\{t_3 \mid (r_i, c, t_3, t_4, f_j) \in Act\}. \quad (2)$$

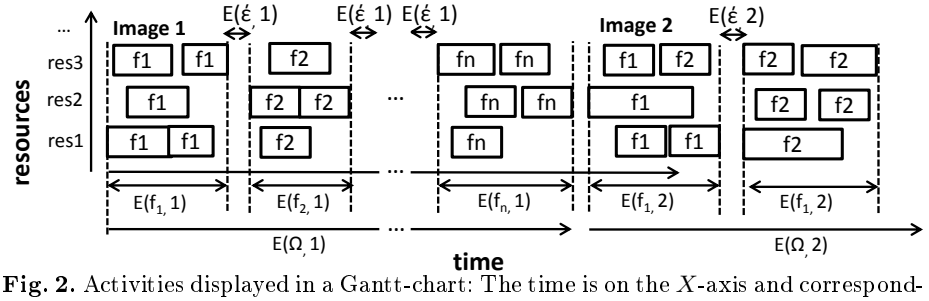

Fig. 2. Activities displayed in a Gantt-chart: The time is on the X-axis and corresponding resources are on the Y-axis. It shows how E activities are grouped into execution times. Activities form rectangles that are labelled with the performed function.

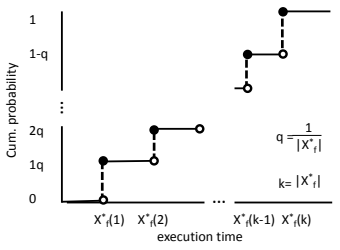

Fig. 3. The empirical distribution function and its inverse, both based on k samples. They are used to determine the probability that a random variable is below a certain value, and for sampling, respectively. It shows the execution time v (X-axis) and corresponding cumulative probability p (Y-axis).

3.3 Using execution times to estimate eCDFs

We now *estimate* eCDFs that summarize execution times for different functions. We group the execution times for function f in an array, where we delete the first j samples from $j+k$ measured cycles to eliminate initial transient behaviour. In order to chose a suitable truncation point j, we use the Conway rule [4], and define j as the smallest integer for each function f that is neither the minimum nor the maximum of the remaining samples:

$$\min(E_f(j+1), ..., E_f(j+k)) \neq E_f(j+1) \neq \max(E_f(j+2), ..., E_f(j+1)).$$

This results in array X_f with $|X_f| = k$ elements, where $X_f(i)$, with $1 \leq i \leq |X_f|$, denotes the i^{th} element of X_f:

$$X_f = (E_f(j+1), E_f(j+2), ..., E_f(j+k)). \tag{3}$$

Now, let X_f^* be a numerically-sorted permutation of X_f, such that $X_f^*(i) \leq X_f^*(j)$, for all $i \leq j$. Clearly, $|X_f^*| = |X_f| = k$ and again, $X_f^*(i)$ with $1 \leq i \leq |X_f^*|$ denotes the i^{th} element of X_f^*.

In the following, we define the eCDF function e_f and its inverse e_f^{-1} based on X_f^*, for all functions f. The eCDF function $e_f(v) : \mathbb{R} \rightarrow [0:1]$ is a discrete, monotonically increasing function that returns the probability that a random variable has a value less than or equal to v. It is defined, for each function f, using the commonly known empirical distribution function [1], as follows:

$$e_f(v) = \frac{1}{k} \sum_{i=1}^{k} \mathbf{1}\{X_f^*(i) \leq v\}, \tag{4}$$

where $\mathbf{1}$ is the usual indicator function. Figure 3 shows an example plot of e_f, based on k values, which consists of $|X_f^*|+1$ horizontal lines, one for each of the cumulative probabilities $(0, q, 2q, 3q, ..., 1)$. It shows that $e_f(v) = \frac{1}{|X_f^*|} = q$, for $X_f^*(1) \leq v < X_f^*(2)$.

The inverse eCDF function $e_f^{-1} : [0:1] \rightarrow \mathbb{R}$ is used to draw samples in line with distribution $e_f(v)$, when simulating. Due to the discontinuities, e_f is not invertible. We resolve this by rounding each probability p to the next

higher probability p' for which $e_f^{-1}(p')$ is defined (see the vertical dotted lines in Figure 3). Thus, $e_f^{-1}(p)$ returns for each $p \in [0:1]$ a value v, as follows:

$$e_f^{-1}(p) = \begin{cases} X_f^*(1), & \text{if } p = 0, \\ X_f^*(\lceil |X_f| \, p \rceil), & \text{if } 0 < p \leq 1. \end{cases} \tag{5}$$

This inverse eCDF $e_f^{-1}(p)$ can be used within the inverse transformation method [5]. Due to the above definition, only actual sample are returned.

3.4 Predicting eCDFs for the complete design space

We now *predict* eCDFs for different designs choices. Formally, a Design Space has n dimensions, each comprising a set of designs alternatives $dim_i = \{val_1, val_2, ..., val_{m_i}\}$, for $1 \leq i \leq n$. The Design Space Model $DSM : dim_1 \times dim_2 \times ... \times dim_n$ is then the n-ary Cartesian product over all dimensions. A Design Space Instance DSI, also called a "design" or "design instance", provides a unique assignment of values to all dimensions: $\overline{x} = (x_1, x_2, ..., x_n)$, where each entry $x_i \in dim_i$ represents the respective design choice for dimension i.

For the sake of simplicity, $Q_{\overline{x}}$ denotes an inverse eCDF e_f^{-1} that is based on a set of measurements of an execution run for design $\overline{x}$. Additionally, $Q_{\overline{x}}(p)$ denotes a sample drawn from $Q_{\overline{x}}$, for probability p.

Clearly the number of designs can grow large, making it costly and infeasible to perform measurements for all possible designs. Hence, we *predict* inverse eCDFs based on other inverse eCDFs without additional measurements, as follows. We carefully select a base design $\bar{b}$ to serve as basis for all eCDF predictions, i.e., $\bar{b}$ is a design that performs well so that its execution times mostly comprise service time and no queueing time. Consequently, set $\hat{Q}$ comprises all inverse eCDFs that need to be acquired through measurements. They correspond to $\bar{b}$ and all neighbours of $\bar{b}$ that differ in exactly one dimension, specified as a union over all dimensions, as follows.

$$\hat{Q} \; = \; \cup_{i=1}^{n} \{ Q_{\bar{b}[v_i]_i} \mid v_i \in dim_i \}, \tag{6}$$

where i is the dimension number, and $\bar{b}[v_i]_i = (b_1, b_2, ..., b_{i-1}, v_i, b_{i+1}, ..., b_n)$.

Let $\bar{t}$ be the design for which the inverse eCDF has to be predicted. We assume that all n design dimensions are independent. As we will see below, this assumption does well in the case we have addressed so far.

Using only inverse eCDFs in $\hat{Q}$, we specify the product of n ratios that each compensate for the difference between $\bar{b}$ and $\bar{t}$ in exactly one dimension:

$$R(p) = \prod_{i=1}^{n} \frac{Q_{\bar{b}[t_i]_i}(p)}{Q_{\bar{b}}(p)}, \tag{7}$$

where $\bar{t}$=(t_1, t_2, ..., t_n), p the probability, and n the number of dimensions.

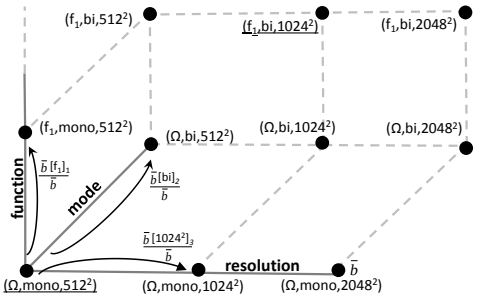

Fig. 4. A geometric interpretation of a 3D Design Space Model; each spatial dimension relates to a design space dimension. Each point in 3D-space represents a Design Space Instance by assigning a value to each dimension. An arrow depicts a ratio between two Design Space Instances.

Measuring all eCDFs in a design space with n dimensions and maximally v values per dimension requires $|DSM| = \mathbb{O}(v^n)$ measurements, while the prediction approach only requires $|\hat{Q}| = \mathbb{O}(vn)$ measurements. Predicting eCDFs is particularly efficient for many dimensions, e.g., for 5 dimensions having 5 values each, prediction requires only 25 out of 3125 (0.8%) eCDFs to be measured.

We illustrate eCDF prediction on an iXR machine with three design dimensions: (i) the image processing function, which is f_1, f_2,..., f_n, or Ω (the sum of all functions); (ii) the mode is either $mono$(plane) for one imaging chain, or bi(plane) for two parallel imaging chains; (iii) the resolution is the number of pixels of the images processed, and is either 512^2, 1024^2 or 2048^2 pixels.

Let $\bar{d} = (f_i, m_j, r_k)$ denote design instance $\bar{d}$ with function f_i, mode m_j, and resolution r_k. It is presented conveniently in 3D-space (see Figure 4). Additionally, $Q_{\bar{d}}$ denotes the inverse eCDF of this particular design.

Let $\bar{t} = (f_1, bi, 1024^2)$ be the design, for which we predict an inverse eCDF. Let $\bar{b}$=$(\Omega, mono, 512^2)$ be the selected base design on which this prediction is based. We then require eCDFs based on measurements for design $\bar{b}$ and for $(f_1, mono, 512^2)$, $(\Omega, bi, 512^2)$ and $(\Omega, mono, 1024^2)$ that each differ from $\bar{b}$ in exactly one dimension and from $\bar{t}$ in all other dimensions. We assume that the three design dimensions are independent. $R(p)$ is then the product of three ratios that each compensate for the difference between design $\hat{b}$ and $\hat{t}$ in one dimension:

$$R(p) \;=\; \frac{Q_{f_1,mono,512^2}(p)}{Q_{\bar{b}}(p)} \;\cdot\; \frac{Q_{\Omega,bi,512^2}(p)}{Q_{\bar{b}}(p)} \;\cdot\; \frac{Q_{\Omega,mono,1024^2}(p)}{Q_{\bar{b}}(p)}. \qquad (8)$$

The eCDF of the design $Q_{\bar{t}}$ is then predicted as follows: $Q_{\bar{t}}(p) \approx Q_{\bar{b}}(p) \cdot R(p)$, for probabilities $p \in [0:1]$. To validate, we compare $R(p)$ with ratio $Q_{\bar{t}}(p)/Q_{\bar{b}}(p)$ that is obtained when measuring $Q_{\bar{t}}(p)$ in Figure 5, for all probabilities $p \in [0:1]$. Figure 5 shows the three ratio terms of (8):

(i) $Q_{(f_1,mono,512^2)}$ / $Q_{\bar{b}}$ (dark blue) compares the execution times of function f_1 and Ω. Function f_1 takes about 0.4 of the total execution time;

(ii) $Q_{(\Omega,bi,512^2)}$ / $Q_{\bar{b}}$ (red) compares the performance of a mono and biplane system. Most values are close to 1. Hence, their performance is comparable;

(iii) $Q_{(\Omega,mono,1024^2)}$ / $Q_{\bar{b}}$ (purple) shows the performance effect of a resolution increase from 512^2 to 1024^2 pixels, which is 3.2 for most probabilities p, which is less than the fourfold increase of pixels.

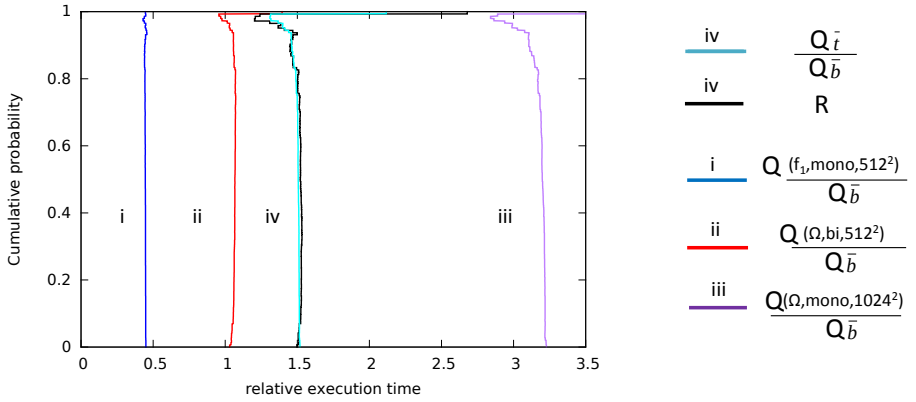

Fig. 5. Inverse eCDFs with relative execution times, which are the quotient of two eCDFs. On the X-axis, it shows relative execution times, and on the Y-axis, cumulative probabilities. Both axes show ratios and are therefore unitless.

Presumably, image processing comprises a constant and pixel dependent part, leading to relatively faster processing for larger images. We also see that (iv) $R(p)$ matches its measurement-based counterpart $Q_{\bar{t}}(p)/Q_{\bar{b}}(p)$ well.

The shown graphs are fairly constant for most probabilities p, which indicates that design instances are linearly dependent. However, they display smaller values for probabilities p close to 1. This is because of the inverse eCDF $Q_{\bar{b}}$, which has high execution times for probabilities near 1. Since all ratios discussed have $Q_{\bar{b}}$ in their numerator, they consequently display smaller values for the same probabilities. In Section 5, we show the results of predicting the performance of designs, using these ratios.

4 Extending the iDSL language & solution chain

In this section, we explain how we use iDSL [13] to automate the solution chain of Figure 1. For this purpose, we have build on previous work of iDSL in which a language and toolbox for performance evaluation of service systems has been constructed. The language comprises six sections that constitute the conceptual model, i.e., Process, Resource, System, Scenario, Measures and Study (see Figure 7). Figure 6 shows the iDSL solution chain that automates the methodology. To support it, the iDSL toolbox has been extended with functionalities “Resolve eCDF", realizing the concepts of Section 3, and “Compute aggregate" (see Figure 6, component 1 and 3). Below, we discuss the iDSL model of iXR systems (Section 4.1), followed by the two extensions (Section 4.2 and 4.3).

4.1 The iDSL model of iXR systems

The iDSL model is defined as follows. *Process* Image Processing (IP) encompasses two high-level functions “Noise reduction" and “Refine", which in turn

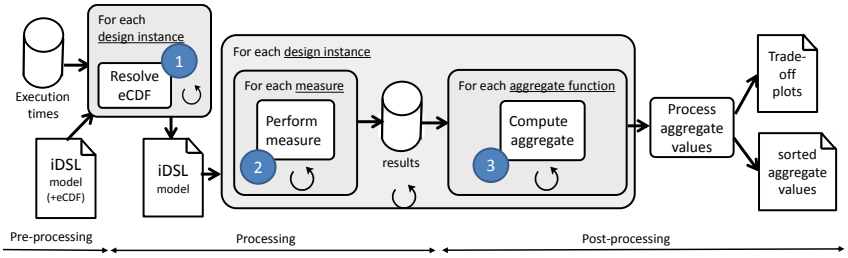

Fig. 6. The fully automated iDSL solution chain. An iDSL model and execution times are used to predict eCDFs, leading to an iDSL model having the predicted eCDFs incorporated in it. For each design, measures are performed and a number of aggregate functions are computed using these measures. Finally, the aggregate values of all design instance are sorted and turned into trade-off plots.

decompose into a sequence of n atomic functions f_1,f_2,...,f_n (as depicted in Figure 8). IP is enclosed by a **Mut**ual **Ex**clusion to enforce a strict FIFO scheduling policy by processing images in one go. The only *resource*, CPU, is equipped with a FIFO queue. *Service* IP maps all processes to the CPU. The *Scenario* prescribes that images arrive f times per second with fixed inter-arrival times, where f is the frame-rate. The *Measure* simulation yields, in one run, latencies of 50 images and the utilization of the CPU. The *Design space* is the Cartesian product resolution and mode. However, to compute two trade-off graphs (as in Figure 9), we also vary the buffer size and frame-rate.

The iDSL model of iXR systems can be depicted as a Discrete-time Markov chain (DTMC) informally, as follows. Its *states* are composed of an image counter, a function counter, the accrued service time of the processed function, the time until the next image arrives, and the queue of Resource CPU. Its key events are the arrival of a new image, which is placed in the queue or discarded when the queue is full, and the current function finishing processing. All states have one or two outgoing *transitions*. The latter case represents a binary probabilistic choice, driven by an eCDF, to decide whether the currently processed function receives

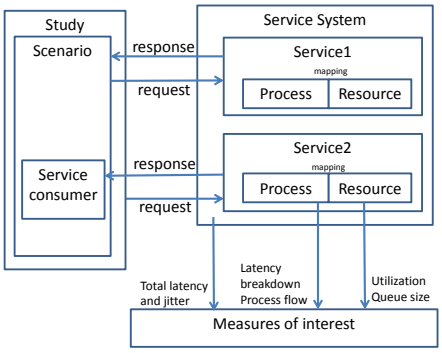

Fig. 7. The concepts of iDSL [13]: A **service system** provides services to consumers. A **service** is implemented using a **process**, **resource** and mapping. A **process** decomposes service requests into atomic tasks, each assigned to resources by the mapping. Resources perform one atomic task at a time. A **scenario** comprises invoked service requests over time. A **study** evaluates a set scenarios to derive the system's characteristics. **Measures** of interest are the retrieved measures.

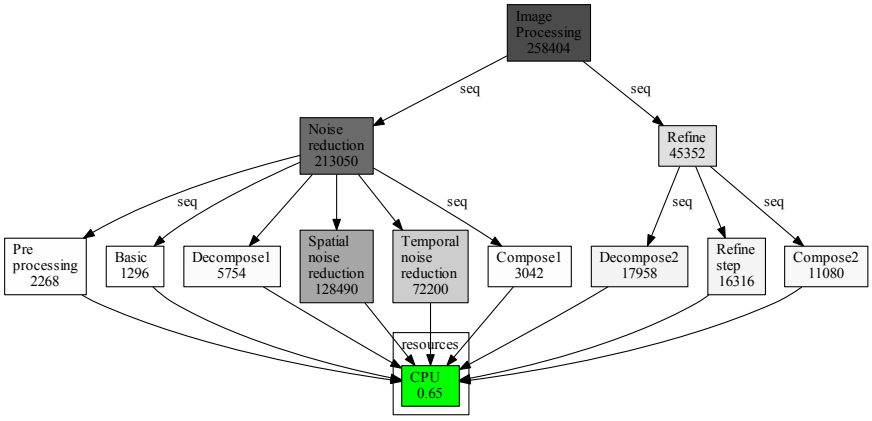

Fig. 8. Service IP contains process IP, which decomposes into a sequential hierarchy of functions. All atomic functions map to resource CPU. In the figure, it shows latency times (in μs) for each function and the utilization for Resource CPU. Latency demanding functions are dark to be easily pinpointed by the system designer. Green Resource CPU has a low utilization. This visual is auto-generated from the iDSL description.

further processing, or finishes. Principally, the number of states is infinite due to an ever-increasing image counter. However, omitting this counter from the state, yields a finite-state DTMC that can be analyzed via model checking [13] to retrieve aggregated latencies, e.g., the maximum latency. iDSL generates the DTMC via the Modest language, which in turn transforms into it a Stochastic Timed Automata [8] network. We only use simulations in this paper.

4.2 Automated prediction of eCDFs for the complete design space

Pre-processing step "Resolve eCDF" performs iDSL model transformations in which eCDF constructs are resolved, using execution times of designs. Our iDSL model (as in Figure 8) contains Process IP with n functions (f_1,f_2,...,f_n) whose probabilistic execution times are individually computed as in (8), for each design. Concretely, "Resolve eCDF" predicts eCDFs using execution times, followed by a discretization step that turns these eCDFs into finite probabilistic choices. It thereby applies the following four steps, for each function, mode and resolution.

First, the required eCDFs, as in the right hand of (8), are obtained by retrieving the corresponding execution times, from which in turn eCDFs are estimated. Second, solving this equation yields the eCDF to be predicted. Third, n samples are taken from this predicted eCDF for probabilities ($\frac{1}{n}$,$\frac{2}{n}$,$\cdots$,1) to discretize it, i.e., we use $n = 1000$. Fourth, these n samples are combined in a probabilistic choice, a process algebra construct that iDSL supports by default. After this, the resulting probabilistic choices are ordered by design and function within that design, and added to the iDSL model.

4.3 Automated aggregation of latencies

The post-processing step "Compute aggregate" applies, for each design, a number of aggregate functions on n obtained latencies from simulations (we use $n = 50$).

We selected the *average*, *maximum* and *median* as the functions of interest (for an example, see Table 1). Concretely, "Compute aggregate" executes when a simulation run finishes and computes the specified aggregate functions.

Next, "Process aggregate values" generates trade-off plots [3, 6] that help the system designer with balancing between two system aspects by plotting these aspects of designs in a 2D-plane (for examples, see Figure 9). They visualize how gains on one system aspect pay its toll on another. A design dominates another design when it ranks better on one aspect and is at least a good on the other aspect. Dominated designs are called Pareto suboptimal, others Pareto optimal.

Finally, "Process aggregate values" sorts design instances on each individual aspect. This enables the comparison of designs on a particular system aspect.

5 Results of a case study on iXR systems

In this section, we study the performance results of an iXR system to show the validity and applicability of our work by evaluating a concrete iXR system.

We obtained all results by executing the constructed iDSL model on a PC (AMD A6-3400M, 8Gb RAM) using 32'27" (minutes, seconds). Predicting eCDFs took 1'48" (6%), simulations 30'13" (91%) and aggregate functions 19" (1%).

5.1 The performance of an iXR system

In the following, we present eCDFs with execution times and corresponding aggregate metrics, a latency break-down graph, and two trade-off graphs.

We assess if the performance of biplane iXR systems on shared hardware is as good as monoplane ones. We use **eCDFs with execution times** in which we compare the measured performance (in Figure 10) of these biplane systems (green) with monoplane ones (red), for resolutions 512^2 (top), 1024^2 (middle) and 2048^2 (bottom). As an effect of sharing hardware, biplane systems perform worse than monoplane ones for image resolutions 512^2 and 1024^2, viz., their average latencies are 6% and 2% higher (as in Table 1), respectively. In contrast, biplane systems with an image resolution of 2048^2 perform 9% better than their monoplane counterparts, due to more powerful hardware biplane systems entail.

Additionally, we assess whether the predictions reflect reality. Therefore, we compare the predicted performance (in Figure 10) of these biplane systems (blue) with the monoplane one (black). They are consistently 6-7% slower, just as the difference in the average and median latency of the **aggregate metrics** (in Table 1). This difference stems from eCDF prediction, i.e., it is the ratio for mode between monoplane and biplane: $Q_{(\Omega,bi,512^2)}$ / $Q_{\bar{b}}$ (in Figure 5).

Furthermore, iDSL creates **latency breakdown charts** for all designs, including design $(\Omega, 2048, bi)$ (see Figure 8). "Spatial noise reduction" and "Temporal noise reduction" (dark gray) are on average the most time consuming functions of IP. We consider the utilization of CPU of 0.65 "good" (green).

Finally, we show two **trade-off graphs** (in Figure 9). They provide insight in how an increase in one system aspect implies a loss in another one. For illustration purpose, the design space is therefore expanded here with the dimensions

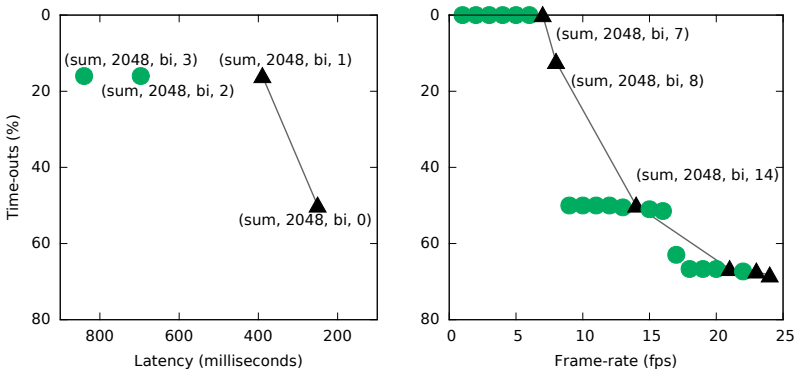

Fig. 9. Two trade-off graphs. Left, designs $(\Omega, 2048, bi, b)$ where b is the buffer size, which affects the relative number of time-outs (y-axis) and the average latency (x-axis). Designs (Ω, 2048, bi, 0) and (Ω, 2048, bi, 1) are Pareto optimal (black triangles), opposed to the other designs that are Pareto suboptimal (green circles). Right, designs $(\Omega, 2048, bi, f)$ where f is the frame-rate (x-axis) affecting the time-out ratio (y-axis).

buffer size and frame-rate. First, Figure 9, (left) shows how the buffer size influences both the average latency (x-axis) and the time-out ratio (the relative amount of images rejected by the system due to overuse) (y-axis), for designs $(\Omega, 2048, bi, b)$ where $b \geq 0$ is the buffer size. Note that both axis are reversed, so that designs that are on the top-right in the graph are preferable, e.g., design $(\Omega, 2048, bi, 1)$ is preferred to $(\Omega, 2048, bi, 2)$. The design with $n = 0$ yields 50% time-outs and a latency of 119ms, whereas the design with $n = 1$ leads to 16% time-outs, but at the price of a latency of 184ms. All designs with $n \geq 2$ yield 16% time-outs, but with an ever increasing latency due to queuing time as n increases, making them Pareto suboptimal.

Figure 9 shows that the frame-rate and the time-out ratio are positively correlated, for designs $(\Omega, 2048, bi, f)$ where f is the frame-rate. For $f \leq 7$, no time-outs occur, whereas for $f > 7$ the number of time-outs increases steadily.

5.2 The validity and applicability of the iDSL model

We compare the predicted and measured eCDFs of six designs to see if the predicted results reflect reality, using two similarity functions for eCDFs: the Kolmogorov distance Kd and the maximum execution ratio Er. Er is inspired by Kd, but returns the "horizontal distance"; these distances are also indicated in the plot in Figure 10. Er is normalized using the median values of its arguments, making Er symmetric and unitless. They are defined as follows.

$$Kd_{m,n} = \sup_{x \in \mathbb{R}} |F_m(x) - F_n(x)|, \qquad Er_{m,n} = \frac{\sup_{p \in [0:1]} |G_m(p) - G_n(p)|}{\frac{1}{2}G_m(0.5) + \frac{1}{2}G_n(0.5)},$$

where m and n are eCDFs, $F_i(x)$ the probability of eCDF i for value x, and $G_i(p)$ the value of eCDF i for probability p.

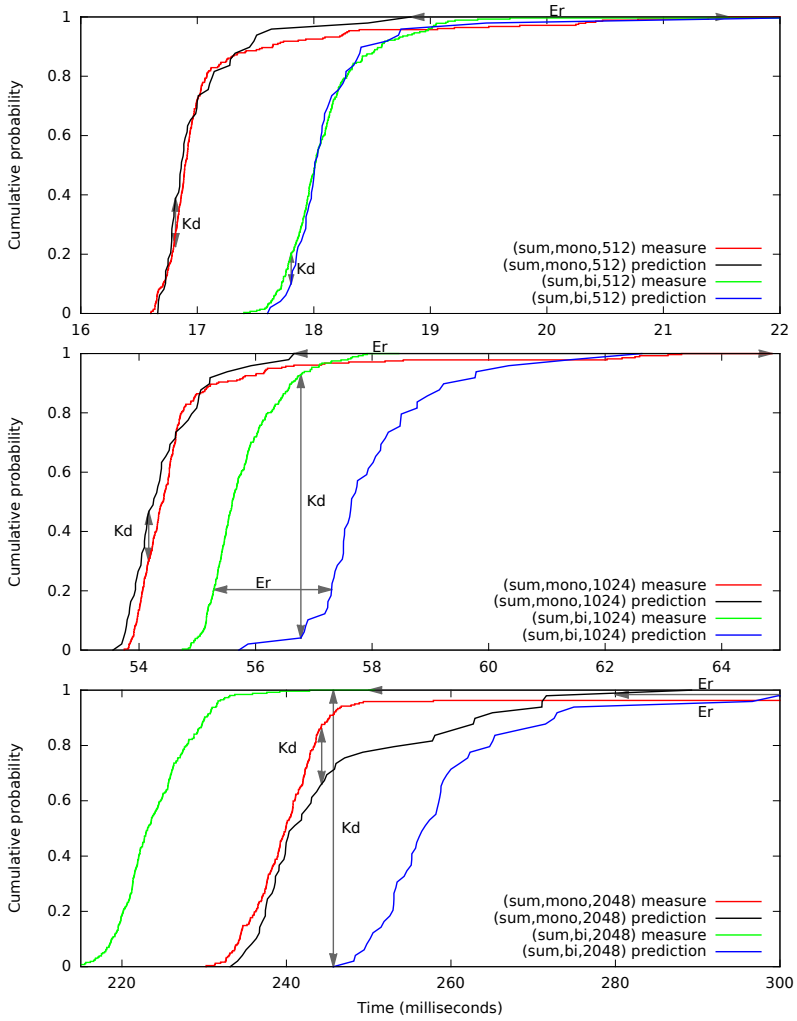

Fig. 10. Measured and predicted execution times eCDFs, for resolution 512 (top), 1024 (middle) and 2048 (bottom), and mode monoplane and biplane. (iDSL auto-generated)

Table 2 shows outcomes for Kd and Er. It shows the maximum distance p and time it occurred for Kd, and the maximum time ratio at which p occurred for Er. Kd is generally low, i.e., most of its values are below 0.16. However, for design $(\Omega, 1024, bi)$ and $(\Omega, 2048, bi)$, Kd is high, 0.86 and 1, resp. Table 1 shows the measured and predicted outcomes of the aggregated functions. Like Kd, predictions for the average and median latency are high for two designs, i.e., for design $(\Omega, 1024, bi)$, 4% and 4%, and for $(\Omega, 2048, bi)$ 16% and 14% difference, resp. Contrarily, the eCDFs for these designs (in Figure 10, green and blue) are not that far apart, although the predictions are clearly conservative. This is due to the relative efficiency gain that occurs when both the resolution and mode are increased, which eCDF prediction does not address.

Table 1. For three aggregate functions, the predicted and measured outcomes (in ms) and their difference Δ, based on the first 50 latency values, for six designs.

	Average latency			**Maximum latency**			**Median latency**		
Design	Pred.	Meas.	Δ	Pred.	Meas.	Δ	Pred.	Meas.	Δ
(Ω, 512, mono)	9	9	0%	9	12	-22%	9	9	0%
(Ω, 512, bi)	9	9	0%	12	15	-21%	9	9	0%
(Ω, 1024, mono)	27	28	-2%	29	33	-12%	27	27	0%
(Ω, 1024, bi)	29	28	4%	30	29	3%	29	28	4%
(Ω, 2048, mono)	122	123	-1%	149	198	-25%	120	120	0%
(Ω, 2048, bi)	130	112	16%	149	126	19%	128	112	14%

Table 2. Comparing measured and predicted eCDFs via two similarity functions.

Design	**Kd**	**x**	**Er**	**p**
$(\Omega, 512, mono)$	0.13	8.4 ms	0.27	1.00
$(\Omega, 512, bi)$	0.08	8.9 ms	0.35	1.00
$(\Omega, 1024, mono)$	0.22	27.0 ms	0.14	1.00
$(\Omega, 1024, bi)$	0.86	28.4 ms	0.04	0.20
$(\Omega, 2048, mono)$	0.15	122.6 ms	0.50	0.98
$(\Omega, 2048, bi)$	1.00	125.1 ms	0.24	0.98

Note that Kd is high when the execution times do not vary much and the overlap is small (in Figure 10, green and blue), while the graphs are fairly similar. Hence, we propose measure Er, tailored to this domain, comparing relative execution times. In the case study, Er has its maximum (see Table 2), for probabilities near 1, the worst case behaviour. Er is high for designs $(\Omega, 512, mono)$, $(\Omega, 512, bi)$ and $(\Omega, 2048, mono)$ due to outliers. However, Figure 10 shows that their graphs are relatively similar, especially for probability values below 0.8.

6 Conclusions & Future Work

In this paper, we used iDSL, a language and toolbox for performance prediction of medical imaging systems. We extended iDSL to support the prediction of unseen eCDFs based on other measured eCDFs, and aggregate functions.

iDSL provides a performance evaluation approach in which we (i) rely on few costly measurements; (ii) use the iDSL toolset to automatically evaluate many designs and present the results visually; (iii) automatically generate aggregated metrics; and (iv) evaluate the performance of complex iXR systems.

In a case study, we have investigated the performance effect of biplane iXR systems on shared hardware. Measurements indicate that these systems perform as good as monoplane ones, but predictions show more conservative results.

iDSL generates latency breakdown charts for each design that show the system designer the process structure, the time consuming processes and resource utilizations, at one glance. iDSL also generates trade-off graphs, in which designs are plotted on two oppose system aspects. They provide the system designer insight in how an increase on one system aspect implies a loss on another one.

We validated the model by comparing its outcomes with measurements. They mostly reflect reality, but are conservative for high resolution biplane systems. The case study involved a medical imaging system, but we consider the approach applicable to many service-oriented systems.

In parallel work we have extended iDSL with probabilistic model checking to obtain execution time eCDFs [12] using the Modest toolset [8]

Acknowledgements We would like to thank Arnd Hartmanns of the Modest team at Saarland University for his efforts made during the development of iDSL, and Mathijs Visser at Philips Healthcare for giving us insight in the iXR system case study and for performing measurements.

References

1. M. Ayer, D. Brunk, G. Ewing, W. Reid, and E. Silverman. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, 26(4):641–647, 1955.
2. T. Basten, E. Van Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. De Smet, L. Somers, and E. Teeselink. Model-driven design-space exploration for embedded systems: the Octopus toolset. In *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6415 of *LCNS*, pages 90–105. Springer, 2010.
3. Y. Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.
4. R. Conway. Some tactical problems in digital simulation. *Management Science*, 10(1):47–61, 1963.
5. L. Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th Winter simulation conference*, pages 260–265. ACM, 1986.
6. R. Ghodsi, M. Skandari, M. Allahverdiloo, and S. Iranmanesh. A new practical model to trade-off time, cost, and quality of a project. *Australian Journal of Basic and Applied Sciences*, 3(4):3741–3756, 2009.
7. J. Groote, A. Osaiweran, and J. Wesselius. Analyzing the effects of formal methods on the development of industrial control software. In *Software Maintenance 27th IEEE International Conference on*, pages 467–472. IEEE, 2011.
8. A. Hartmanns and H. Hermanns. The modest toolset: An integrated environment for quantitative modelling and verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LCNS*, pages 593–598. Springer, 2014.
9. S. Haveman, G. Bonnema, and F. van den Berg. Early insight in systems design through modeling and simulation. *Procedia Computer Science*, 28:171–178, 2014.
10. G. Igna and F. Vaandrager. Verification of printer datapaths using timed automata. In *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *LCNS*, pages 412–423. Springer, 2010.
11. J. Johnson. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*. Morgan Kaufmann, 2010.
12. F. van den Berg, J. Hooman, A. Hartmanns, B.R. Haverkort, and A. Remke. Computing response time distributions using iterative probablistic model checking. In *Computer Performance Engineering*, this volume of LCNS. Springer, 2015.
13. F. van den Berg, A. Remke, and B. R. Haverkort. A domain specific language for performance evaluation of medical imaging systems. In *5th Workshop on Medical Cyber-Physical Systems*, pages 80–93. Schloss Dagstuhl, 2014.
14. F. van den Berg, A. Remke, A. Mooij, and B.R. Haverkort. Performance evaluation for collision prevention based on a domain specific language. In *Computer Performance Engineering*, volume 8168 of *LCNS*, pages 276–287. Springer, 2013.
15. J. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430–437, 2003.