

Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Design Space Exploration of the Quality of Service for Stream Reasoning Applications

av

Viet Ha Nguyen

LIU-IDA/LITH-EX-A--12/027--SE

2012-08-09



Linköpings universitet



På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

© Viet Ha Nguyen

Final thesis

**Design Space Exploration of the
Quality of Service for Stream
Reasoning Applications**

by

Viet Ha Nguyen

LIU-IDA/LITH-EX-A-12/027-SE

August 9, 2012

Final thesis

Design Space Exploration of the Quality of Service for Stream Reasoning Applications

by

Viet Ha Nguyen

LIU-IDA/LITH-EX-A-12/027-SE

August 9, 2012

Supervisor: **Dr. Unmesh D. Bordoloi**
Dr. Fredrik Heintz

Examiner: **Prof. Dr. Zebo Peng**

Abstract

An Unmanned Aerial Vehicle (UAV) is often an aircraft with no crew that can fly independently by a preprogrammed plan, or by remote control. Several UAV applications, like autonomously surveillance and traffic monitoring, are real-time applications. Hence tasks in these applications must complete within specified deadlines.

Real Time Calculus (RTC) is a formal framework for reasoning about real-time systems and in particular streaming applications. RTC has its mathematical roots in Network Calculus. It supports timing analysis, estimating loads and predicting memory requirements.

In this thesis, a formal analysis of real-time stream reasoning for UAV applications is conducted. The performance analysis is based on RTC using an abstract performance model of the streaming reasoning on board a UAV. In this study, we consider two different scheduling methods, first-in-first-out (FIFO) and fixed priority (FP). In the FIFO scheduling model the priorities of the tasks are assigned and processed based on the order of their arrival, while in the FP scheduling model the priorities of the tasks are preassigned. The Quality of Service (QoS) of these applications is calculated and analyzed in a proposed design space exploration framework.

QoS can be defined differently depending on what field we are studying and in this thesis we are interested in studying the delays of the real-time stream reasoning applications when (i) we fix jitters and number of instances and vary the periods, (ii) we fix the periods and number of instances and vary the jitters, and (iii) we fix the periods, jitters and vary the number of instances.

Acknowledgements

I would like to express my gratitude towards my examiner, Prof. Dr. Zebo Peng and my two supervisors, Dr. Unmesh D. Bordoloi and Dr. Fredrik Heintz for providing me an opportunity to work on this project. I would like to thank both my supervisors for supporting me with ideas and guidance throughout the thesis. Last but not least, my family for always supporting and encouraging me with their best wishes.

Contents

1	Introduction	1
1.1	Problem definition	2
1.2	Motivation	3
1.3	Related Works	3
1.4	Organization	4
2	Real Time Calculus	5
2.1	Events - Arrival curves	5
2.2	Resources - Service curves	7
2.3	Computing delays	7
2.4	RTC implementations	8
2.4.1	Events	8
2.4.2	Resources	9
2.4.3	Scheduling methods	11
3	A design space exploration framework	13
3.1	The performance model	16
3.1.1	FIFO model	16
3.1.2	FP model	18
4	Case study	19
4.1	Experimental setup	19
4.2	First in first out analysis	20
4.3	Fixed priority analysis	23
5	Conclusion	33
	Bibliography	34

List of Figures

1.1	A typical real-time system.	2
2.1	Events in time domain and RTC.	6
2.2	Parameter definitions for events in time domain.	6
2.3	Resource in time domain and RTC.	7
2.4	The maximum delay of an event.	8
2.5	Parameter definitions for events in RTC.	8
2.6	Different events defined in RTC.	9
2.7	Different resources defined in RTC.	10
3.1	Design space exploration steps.	13
3.2	Block view of the suggested design space exploration framework.	14
3.3	Example on how combinations of different task parameters are arranged.	15
3.4	Block view of the suggested FIFO model.	17
3.5	Block view of the suggested FP model.	18
4.1	Delay for task 3, with period P_3 fixed at (a) 10 ms and (b) 15 ms.	24
4.2	Delay for task 3, with period P_3 fixed at (a) 20 ms and (b) 25 ms.	25
4.3	Delay for task 3, with period P_3 fixed at (a) 30 ms and (b) 35 ms.	25
4.4	Delay for task 3, with period P_3 fixed at (a) 40 ms and (b) 45 ms.	26
4.5	Delay for task 3, with period P_3 fixed at (a) 50 ms and (b) 55 ms.	26
4.6	Delay for task 3, with P_3 period fixed at 40 ms.	27
4.7	Delay of four tasks with period variation in the range 5-70 ms.	27
4.8	Delay for task 3, with jitter fixed at 10 ms in 3D-view.	29
4.9	Delay of four tasks with jitter variation in the range 0-70 ms.	30
4.10	Delay of four tasks with instance variation in the range 1-300.	31
4.11	Delay of task 3 with fixed number of instances in 3D view.	32

List of Tables

4.1	Default task information for First-in-First-out scheduling. . .	20
4.2	Default task information for Fixed-Priority scheduling.	20
4.3	Task set when varying the number of instances.	21
4.4	Task set when varying the periods.	21
4.5	Task set when varying the jitters.	21
4.6	Task set when varying the periods.	23
4.7	Task set when varying the jitters.	29
4.8	Task set when varying the number of instances.	31

Chapter 1

Introduction

An Unmanned Aerial Vehicle (UAV) is usually an aircraft with no pilot on board. UAVs can be controlled remotely by pilots on the ground or they can fly independently based on some preprogrammed plans. We are mainly interested in autonomous UAVs which are UAVs with little or no involvement of a pilot on the ground. Most existing UAV applications today are basically remote controlled aircraft. They are widely used in many applications, for instance, delivering supplies to potentially dangerous areas and scanning urban areas for traffic violations or criminal activities. On board the UAV there is an embedded real-time system which often has well-defined, fixed-time constraints. Real-time systems in general are often associated with deadlines therefore reasoning must be done within the defined deadlines or the system will fail. Broadly, real-time systems can be classified into two types: soft real-time systems and hard real-time systems.

- In a soft real-time system, if some deadlines are missed, these will not lead to severe consequences.
- In a hard real-time system, a missed deadline would result in a catastrophic failure of the system.

Assume a UAV [1] is given a task of monitoring a small urban area for potential or ongoing criminal activities. In order to reason about the current activities it is necessary for the UAV to create a world model and maintain the correspondence between this world model and the real world. For traffic monitoring this involves, for example, creating symbolic representations of cars. One approach to acquiring such a symbolic representation is to analyze video sequence taken by on-board cameras. The analysis could be based on an image-based tracking system together with some object classification and anchoring algorithms.

In a real-time system such as a UAV, there may be stream reasoning applications which are modeled and implemented using the DyKnow [1, 2]

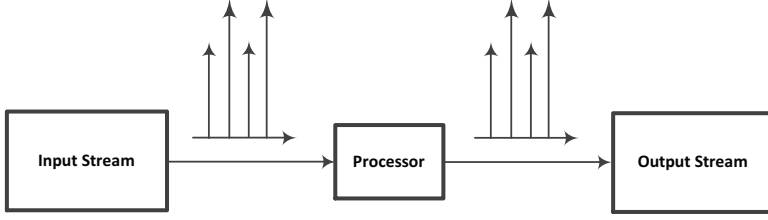


Figure 1.1: A typical real-time system.

stream-based knowledge reasoning middleware. Dyknow models a streaming application as a set of computational units connected by streams. A computational unit represents a computation which takes one or more streams as inputs and produces one or more streams as output. Each stream is associated with a policy which is a declarative representation of its desired properties. Figure 1.1 shows how a typical real-time system which contain input streams, reasoning units and output streams. An example application could be reasoning over video streams to detect traffic violations for a UAV monitoring an urban area. If the application is missing its deadline it can lead to unwanted results such as traffic violations that are not detected. Therefore we need to provide Quality of Service (QoS) guarantees for such systems. QoS can be defined differently depending on what field we are studying. For instance, in a shop the QoS can be customer satisfactions or the different variety of products that the shop can offer and in a restaurant, the QoS can be delivery time of the meals to the customers. In real-time systems, there are many ways to measure the QoS, for example buffer requirements, delays, CPU usage etc. and in this work we are interested in studying how the delays of the stream reasoning applications behave because the timing behavior for these streaming applications are not yet known.

1.1 Problem definition

The stream reasoning applications are running on a uniprocessor environment. Let $\tau = \{T_1, T_2, \dots, T_n\}$ denotes the set of stream reasoning applications running on this processor. Assuming that for each task T_i , the period is P_i , the jitter is J_i , minimum inter-arrival distance D_i where $i = 1 \dots n$. We also assume that the best-case and the worst-case execution times of a task T_i are given as $BCET_i$ and $WCET_i$, respectively, where $i = 1 \dots n$. There is also an assumption that there can be more than one instance of a task to arrive at the same moment. Here, the number of instance means the number of copies of a certain task, where these instances have the same period, jitter and minimum inter-arrival distance.

1. The problem is to develop a methodology for deriving a RTC performance model, given a particular DyKnow stream application and

a specific system specification. The system specification includes the processor information as well as scheduling policies to be used.

2. How the RTC performance models can be used for design space exploration of stream reasoning applications modeled using DyKnow. We consider the following case when doing design space exploration.
 - (a) The dependence of the QoS on the number of instances with fixed periods and jitters.
 - (b) The dependence of the QoS on the jitters with fixed periods and number of instances.
 - (c) The dependence of the QoS on the periods with fixed jitters and number of instances.

1.2 Motivation

One way to estimate the QoS is to conduct simulations. Computer based simulation techniques can only model and simulate the behavior of a system but cannot provide guarantees because these techniques can only simulate a small part of the possible behaviors of a system. Moreover this method is time consuming and it is costly to build and maintain complex models. Therefore, it is necessary to perform a formal analysis of the system. This can be done by doing a formal analysis of the whole system. The theory used for the formal analysis is Real Time Calculus (RTC). This can be done by using an abstract performance model by applying the RTC theory to determine (i) when the system functions properly and (ii) the border where the system may potentially have failures.

1.3 Related Works

RTC [3, 4] has been used as a general framework for analyzing different system properties with respect to timing analysis, loads on various components and on-chip buffer memory requirements of heterogeneous platform-based architectures, in a single coherent way. The framework is shown to be capable of formal analysis of different system properties in heterogeneous platform-based design using different scheduling policies.

Wandeler and Thiele [5, 6] have applied RTC to different modular performance analysis in an actual case study on an in-car radio navigation system with several system architectures. The analysis based on the model is efficient and suitable for early design space exploration due to its high level abstraction. By the case study of an actual in-car radio system the authors showed that the model is also suitable for modeling control-oriented and distributed software-intensive systems. It has been shown that the

abstract functionality models can be combined to modules for modular performance analysis of an embedded system. RTC has also been used during design space exploration as presented in [6]. In [7] Henia et al. proposed a design space exploration tool flow based on SymTA/S. The tool supports heterogeneous architectures, complex task dependencies and context aware analysis. It determines system-level performance data such as end-to-end latencies, bus and processor utilization, and worst-case scheduling scenarios. SymTA/S furthermore combines optimization algorithms with system sensitivity analysis for rapid design space exploration [7].

On board the UAV [1] there are several stream reasoning applications modeled using the DyKnow stream-based knowledge processing middleware [2]. The system uses metric temporal logic and chronic recognition to detect complex situations such as take overs and intersection behaviors. Because the UAV is a real-time embedded system and the computational resources are limited, there is a need to analyze how to best use these resources in order to guarantee the QoS of the applications.

1.4 Organization

The thesis is organized as follows. Chapter 1 gives a brief introduction on the objective and methods to be used in the study. In Chapter 2, the concepts and theories behind RTC are described. The design space exploration framework is presented in Chapter 3. Chapter 4 provides the results of a case study and Chapter 5 contains the conclusion of this thesis and some future work suggestions.

Chapter 2

Real Time Calculus

When designing embedded real-time systems it is desirable to explore different possibilities as early in the design stage as possible. This requires information from the system level performance analyses. RTC has been widely used as formal analysis framework [3, 4, 5, 6] for analyzing and reasoning about real-time systems, especially streaming applications. RTC is based on Network Calculus [8] and its basic concepts are also extended to the domain of real-time embedded systems. The root of network Calculus is the theory of min-plus and max-plus algebra [4, 6]. The framework supports timing analysis, load estimation and prediction of the memory requirements. RTC requires an abstract performance model. This model contains the necessary information regarding the resources, communication, involved dedicated hardware/software components and also the architecture itself. The input streams will be modeled as arrival curves whereas the resources will be modeled as services curves. How the modeling of events and resources is done, will be described in detail in the next section.

2.1 Events - Arrival curves

Let us define a data stream $R(t)$ as a number of events in the time interval $[0, t)$. Assuming that the events arriving within any time interval is bounded above by a right-continuous, non-negative, sub-additive function called upper arrival curve α^u and lower arrival curve α^l , respectively. $R(t)$, α^u and α^l are related by the following inequality

$$\alpha^l(t - 0) \leq R(t) \leq \alpha^u(t - 0)$$

where $\alpha^l(0) = \alpha^u(0) = 0$ at $t = 0$. This means that between a time interval Δ the minimum and maximum number of events will be $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$, respectively.

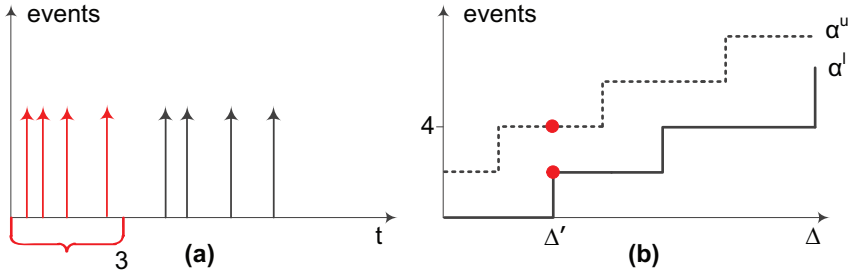


Figure 2.1: Events in time domain and RTC.

In the time domain there are four events arriving within the time of 3 ms as shown in Fig. 2.1a. In RTC the events are modeled into two curves, upper and lower arrival curves, respectively according to Fig. 2.1b. The ring marked places shows how many events are arriving at the time interval Δ' where $\Delta' = 3$. The events are modeled with the common pattern of period, p , jitter, j and minimum inter-arrival distance, d . Here, period is the distance between each event arrival, jitter is the deviation of the true period of a certain event and minimum inter-arrival distance is the distance between each event with jitter. Fig. 2.2a shows purely periodic events with period p while Fig. 2.2b shows events with jitter j and minimum inter-arrival distance d .

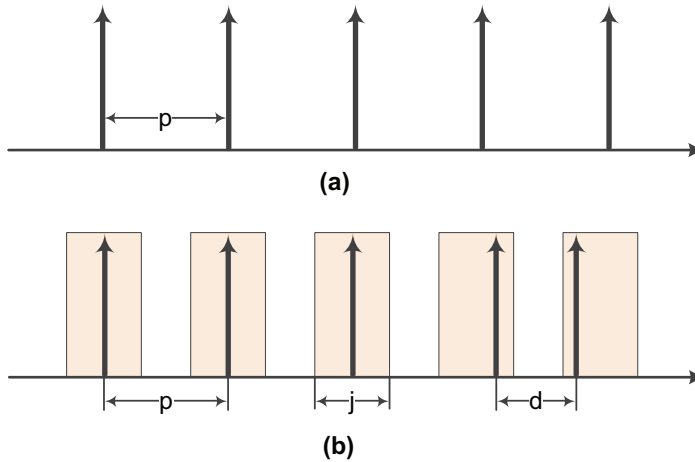


Figure 2.2: Parameter definitions for events in time domain.

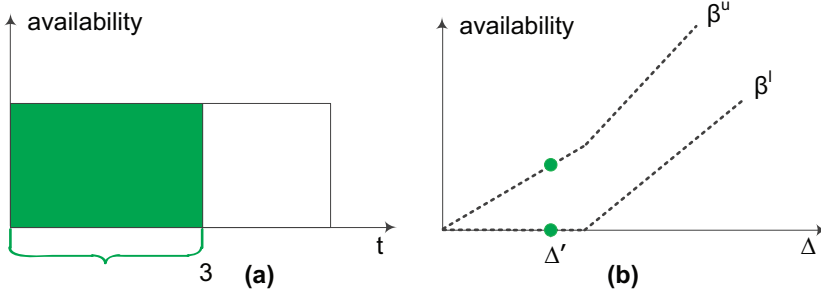


Figure 2.3: Resource in time domain and RTC.

2.2 Resources - Service curves

Similar to the upper and lower arrival curves, we denote the lower and upper service curves as β^l and β^u , respectively. Let $C(t)$ defines the available reasoning power from a certain resource over the time interval $[0, t)$. $C(t)$, β^l and β^u are related by the following inequality

$$\beta^l(t - 0) \leq C(t) \leq \beta^u(t - 0)$$

where $\beta^l(0) = \beta^u(0) = 0$. As shown in Fig.2.3a, the shaded marked area is the available resource of a system between $[0, 3)$ ms while in RTC this resource can be transformed into two curves, upper and lower service curves, respectively as shown in Fig. 2.3b. The ring marked places shown in Fig. 2.3b indicates the minimum and maximum resources at the time interval Δ' where $\Delta' = 3$.

2.3 Computing delays

Assuming a case with a single event that is bounded by an upper arrival curve α^u is processed on a resource bounded by a lower service curve β^l , then the maximum delay D (see Fig. 2.4) experienced by this particular event can be found by applying the inequality

$$D \leq \sup_{\Delta \geq 0} \{ \inf \{ \tau \geq 0 : \alpha^u(\Delta) \leq \beta^l(\Delta + \tau) \} \}$$

The maximum delay is the maximum horizontal distance between the upper arrival curve, α^u and the lower service curve, β^l . When a certain event stream is processed by a various number of components then the maximum delay of each individual component can be summed together, resulting in an end-to-end delay guarantee.

The events and resources can be modeled by applying the RTC theory. This theory is implemented as a Matlab toolbox which allows you to eval-

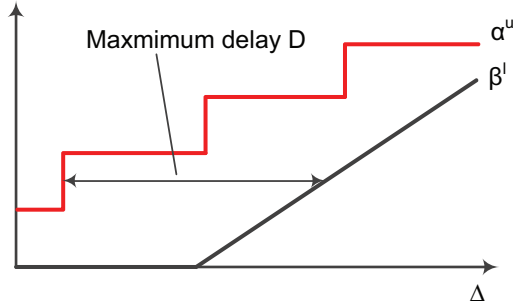


Figure 2.4: The maximum delay of an event.

uate different system properties based on this theory. Some of its specific functions used for this study will be described in the next section.

2.4 RTC implementations

2.4.1 Events

Events in RTC are modeled with the pattern of three parameters: period (p), jitter (j) and minimum inter-arrival distance (d) of an event. Fig. 2.5 shows how different parameters for the RTC function $rtcpjd(p, j, d)$. As can be seen in the figure, if $j = 0$ then the upper and the lower arrival curves represent a purely periodic event. This function models the event according to these three parameters. Fig. 2.6 shows typical examples of arrival curves. The arrival curves in Fig. 2.6a model a strictly periodic event, while the arrival curves in Fig. 2.6b model a periodic event with jitter. Figures 2.6c and 2.6d show a periodic event with bursts and an event stream with more advanced timing behavior, respectively.

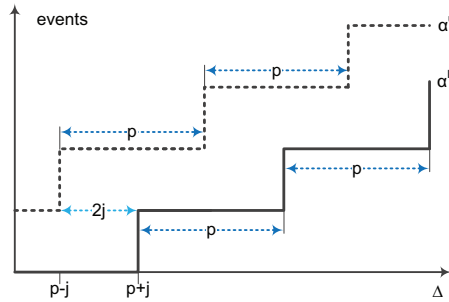


Figure 2.5: Parameter definitions for events in RTC.

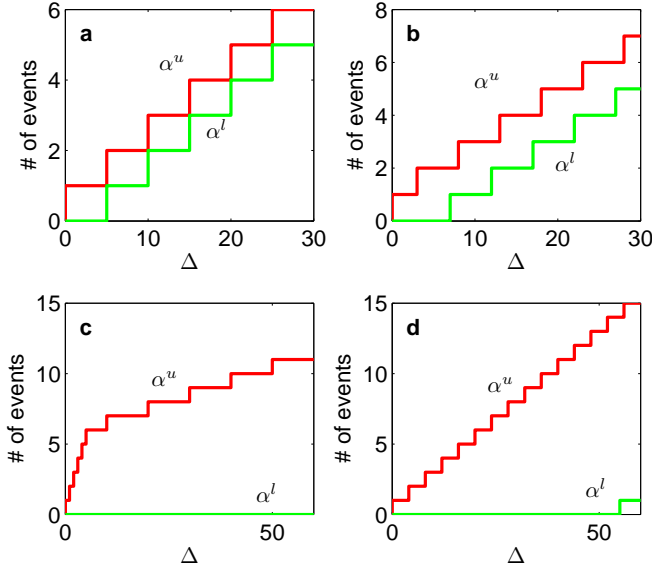


Figure 2.6: Different events defined in RTC.

2.4.2 Resources

There are many functions that model different service curves (Fig. 2.7). Fig. 2.7a shows an example of a full service curve modeled with the RTC function $rtcfs(b)$, where b is the bandwidth. This function will create a full service resource. An arrival task will be guaranteed with this resource as soon as it arrives. Figure 2.7b shows a time division multiple access (TDMA) resource modeled with the function $rtctdma(s, c, b)$ where s is the slot length allocated on a resource with a total bandwidth b and a TDMA cycle c . Fig. 2.7c shows a bounded delay resource which can be created by using the function $rtcbd(d, b)$ where d is the maximum delay and b is the bandwidth. A periodic service can be modeled with this function $rtcps(s, p, b)$ where a share of s is allocated within every period p , on a resource with a total bandwidth of b is shown in Fig. 2.7d.

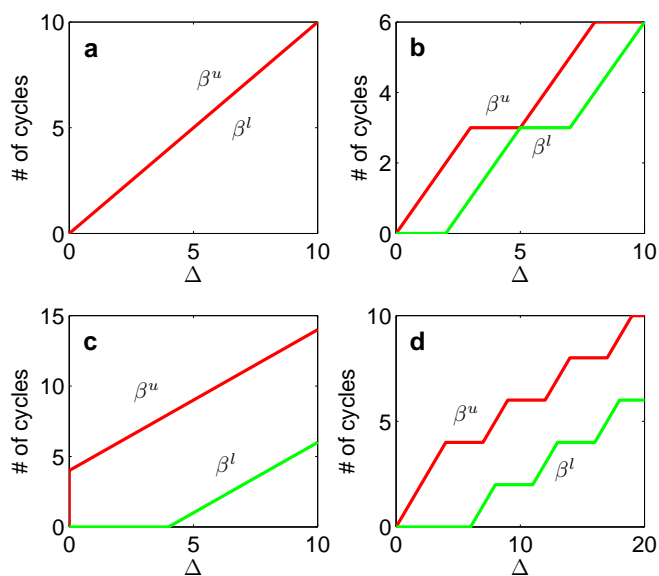


Figure 2.7: Different resources defined in RTC.

2.4.3 Scheduling methods

There are several scheduling policies implemented on-board the UAV such as first-in-first-out (FIFO) and fixed-priority (FP). FIFO is a scheduling policy which is based on the order of arrival of the tasks. The first tasks arrive will be assigned the highest priority while the task arrive last will be assigned lowest priority. FP is also a scheduling policy which is based on preassigned priorities of the tasks. Tasks running with both FIFO and FP policy are preemptive which means if a lower priority task is running and a higher priority task arrives, the higher priority task will be granted access to the resource. Following is a more detailed implementation of the scheduling policies. FIFO and FP are implemented as FIFO [4] component and Greedy Processing Component (GPC) [4], respectively.

First in first out

The output of a first-in-first-out (FIFO) processing component is calculated by the RTC function for the case of two events $rtcfifo(\alpha_1, ed_1, \alpha_2, ed_2, \beta)$, where α_1 and α_2 are the two arrival curves with corresponding execution demands ed_1 and ed_2 , respectively, and the service curve β . In this case the events are processed in the order of their arrival at the component.

$$[\alpha'_1 \text{ del}_1 \text{ buf}_1 \alpha'_2 \text{ del}_2 \text{ buf}_2 \beta'] = rtcfifo(\alpha_1, ed_1, \alpha_2, ed_2, \beta)$$

The function computes the output arrival curves, the worst-case delays for the events, the necessary buffer requirements and the remaining service. Here, the execution demand **ed** is a matrix containing the worst-case and the best-case execution demands [*wced bced*]. The execution demands can be a scalar in this case if $wced = bced$ (worst-case execution demand = best-case execution demand). This function will issue a warning if the set of events is not schedulable.

Fixed priority

For fixed priority (FP) scheduling, an abstract component known as GPC will be used to model the execution of tasks on a single resource. The GPC is a component that is triggered whenever an event is available on the input event stream (described by the arrival curve α) and produces a single output event stream (described by the arrival curve α'). At every event arrival, a task is instantiated to process the incoming event [9]. Considering an example function with two events

$$\begin{aligned} [\alpha'_u \alpha'_l \beta'_u \beta'_l \text{ del}_1 \text{ buf}_1] &= rtcgpc(\alpha_{1_u} \alpha_{1_l} \beta_{1_u} \beta_{1_l} \text{ ed}_1) \\ [\alpha'_u \alpha'_l \beta'_u \beta'_l \text{ del}_2 \text{ buf}_2] &= rtcgpc(\alpha_{2_u} \alpha_{2_l} \beta_{2_u} \beta_{2_l} \text{ ed}_2) \end{aligned}$$

The function will process each event individually and the resource left after reasoning the first event will be available for the next event. The

function *rtcgpc* takes the upper and lower arrival curve α_u, α_l as input, with corresponding execution demand ***ed*** and a service curve β . Here, the matrix ***ed*** also contains two values, worst-case and best-case execution demands [***wced bced***]. As a result, we obtain the output arrival curves, the worst-case delays for the events, the necessary buffer requirements and the remaining service for the next event. The execution demands for FP can be reasoned in the same way as FIFO component.

Chapter 3

A design space exploration framework

Design space exploration is an essential part during a design and there are many iterations over a large space of configurations until the optimal solution is achieved. As shown in Fig. 3.1, we can vary the application data in the Applications block or we can change to different architectures in the Platforms block or even change to different scheduling algorithms in the Scheduling Policies blocks. Each time we change a certain input to one of these three blocks, we will get a new RTC performance model.

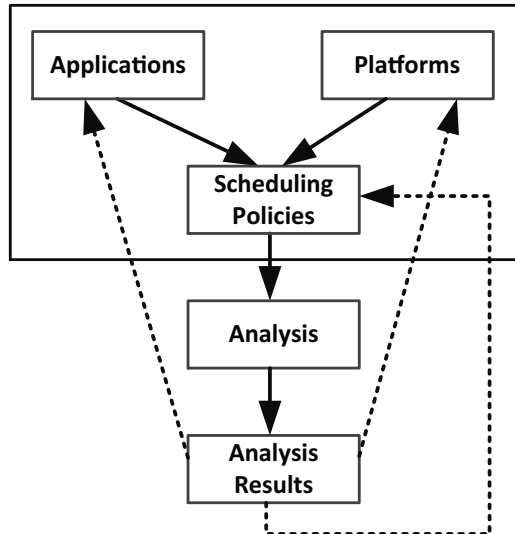


Figure 3.1: Design space exploration steps.

We have to manually edit the inputs to these performance models and

this can be a time consuming task if the data is bounded over a large interval. Therefore we can automate this process by automatically generate all the necessary combination of data and RTC performance models.

Figure 3.2 shows a high level view of the proposed framework that contains four essential parts. The structure of the design space exploration remains the same but there are two new blocks added, Program block and DyKnow application data block. By entering a number as input to the program, different files containing different combinations of the tasks are generated. The number specifies how large range a certain parameter is varied. For each task there are four parameters: period, jitter, minimum inter-arrival distance, WCET and BCET. The RTC performance model contains the following blocks: DyKnow application data, Applications, Platforms and Scheduling Policies. The RTC performance model will be described in more detail in the next section.

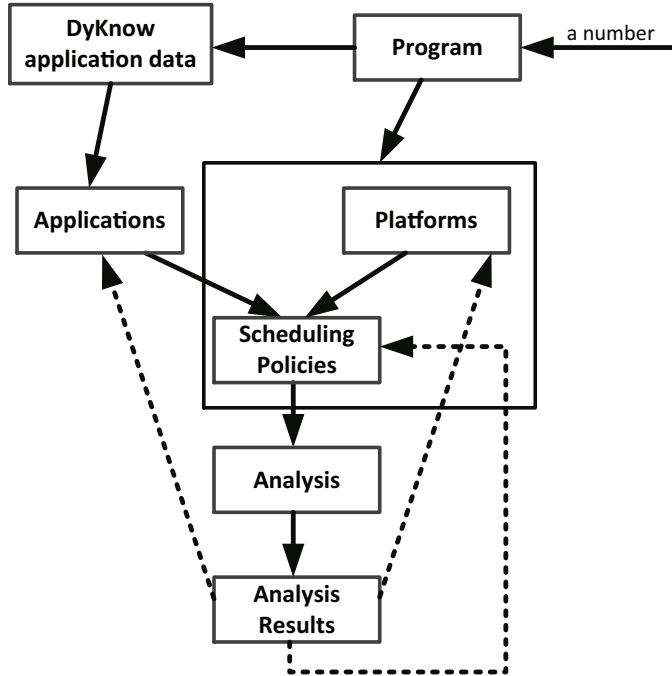


Figure 3.2: Block view of the suggested design space exploration framework.

Algorithm 3.0.1: RTC(n)

```

 $n \leftarrow \text{Input}$ 
for  $i \leftarrow 0$  to  $n$ 
  reset starting point task 2
   $\text{GenT1}(i)$ 
  for  $j \leftarrow 0$  to  $n$ 
    reset starting point task 3
     $\text{GenT2}(j)$ 
    for  $k \leftarrow 0$  to  $n$ 
      reset starting point task 4
       $\text{GenT3}(k)$ 
      for  $l \leftarrow 0$  to  $n$ 
         $\text{GenT4}(l)$ 
   $\text{GenPerfModel}(n)$ 

```

The Program block in Fig. 3.2 contains the program-code that generates all the possible combinations of the task data and the performance models. The program is compacted into the pseudo-code as shown in algorithm 3.0.1. There are four nested loops in algorithm 3.0.1. Each *for* loops will generate data corresponding to each task. At the end of the algorithm, the function $\text{GenPerfModel}(n)$ generates all the performance models corresponding to the case you want to study. Fig. 3.3 shows an example of how the tasks were arranged if we want to have different combinations of the tasks when varying a certain parameter. As shown in Fig. 3.3, the periods are varied according to column one, five and nine.

%period1	jitter1	distance1	wcet1	period2	jitter2	distance2	wcet2	period3	jitter3	distance3	wcet3
5	0	0	0.7	5	0	0	0.6	5	0	0	0.3
5	0	0	0.7	5	0	0	0.6	10	0	0	0.3
5	0	0	0.7	5	0	0	0.6	15	0	0	0.3
5	0	0	0.7	5	0	0	0.6	20	0	0	0.3
5	0	0	0.7	5	0	0	0.6	25	0	0	0.3
5	0	0	0.7	5	0	0	0.6	30	0	0	0.3
5	0	0	0.7	5	0	0	0.6	35	0	0	0.3
5	0	0	0.7	5	0	0	0.6	40	0	0	0.3
5	0	0	0.7	5	0	0	0.6	45	0	0	0.3
5	0	0	0.7	5	0	0	0.6	50	0	0	0.3
5	0	0	0.7	5	0	0	0.6	55	0	0	0.3
5	0	0	0.7	5	0	0	0.6	60	0	0	0.3
5	0	0	0.7	5	0	0	0.6	65	0	0	0.3
5	0	0	0.7	5	0	0	0.6	70	0	0	0.3
5	0	0	0.7	5	0	0	0.6	75	0	0	0.3

Figure 3.3: Example on how combinations of different task parameters are arranged.

3.1 The performance model

In this section, we propose and construct two models based on the scheduling methods FIFO and FP and the problem formulation mentioned at the introduction. Details regarding each model are described later in this chapter.

Algorithm 3.1.1: $[\text{DEL BUF UTIL}] = \text{PERF}(\text{case})$

```

case  $\leftarrow$  Input
switch case
  case = period
    construct the curves
    calculate [del buf util]
    save to file
  case = jitter
    construct the curves
    calculate [del buf util]
    save to file
  case = instance
    construct the curves
    calculate [del buf util]
    save to file

```

We can also have a closer look at Applications, Platforms and Scheduling Policies blocks in Fig. 3.2 which contains the performance models (the RTC code) that were generated by the program in the Program block. The RTC code for the performance analysis can be described by the pseudo-code in algorithm 3.1.1. There will be three different cases for each scheduling methods, variations of the periods, jitters and the number of instances. In each case, the datafile that was generated from the algorithm 3.0.1 will be used to create arrival curves and service curves. The function *rtcfifo* and *rtcgpc* for the calculation of the matrix containing the information of how the delays, buffers and utilization behaves. During the calculation of the matrix $[\text{del}, \text{buf}, \text{util}]$, if any of the values turn out to be infinite, then these values will be automatically converted to a reasonable high number. For the visualization of the data we receive from the experiments, we use two other Matlab toolboxes, Curve Fit and Surface Fit.

3.1.1 FIFO model

Considering a FIFO component with n event streams and one resource as in Fig 3.4. For each set of incoming event streams $\alpha_i, \forall i = 1 \dots n$ to the

FIFO component, $\mathbf{ed}_i, \forall i = 1 \dots n$ is given as a matrix containing the worst-case and best-case execution demands and the service curve β . The event streams are arriving at the component all at the same time but the one that arrives first will get the chance to get processed first. The available resource for all the event streams in this model is the processor. As outputs of this model we obtain three different matrices containing the delays, the buffer requirements and the remaining services.

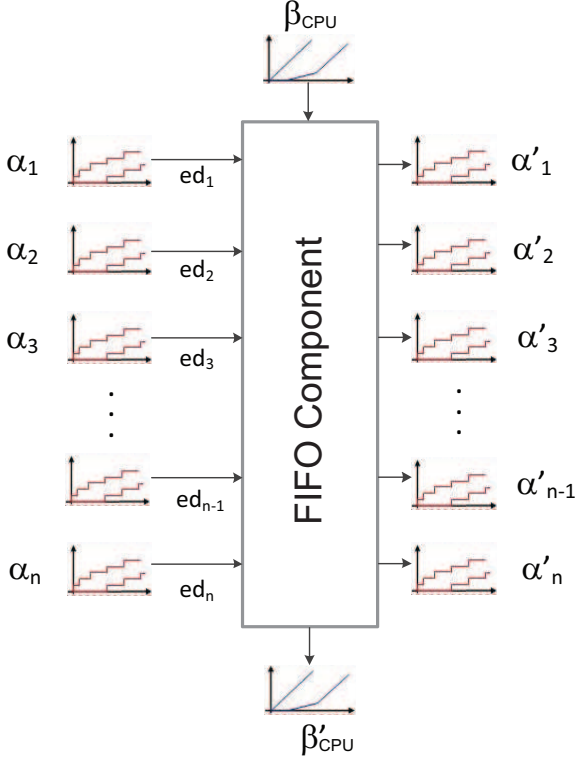


Figure 3.4: Block view of the suggested FIFO model.

3.1.2 FP model

As mentioned in Chapter 2, the FP scheduling is using the greedy processing component [4]. Considering a FP model with n event streams, $\alpha_i, \forall i = 1 \dots n$ and a resource β_1 as shown in Fig 3.5. For each set of incoming event streams, $ed_i, \forall i = 1 \dots n$ is given as a matrix containing worst-case and best-case execution demands. Compared to the previous model, the event streams in this model are processed in a greedy manner, that means for example, the event stream α_1 has the highest priority and will have full access to the resource β_{CPU} . The remaining resource, β'_1 , is the available resource for the incoming event stream α_2 when the event stream α_1 finishes. This process continues until all event streams are processed. This may also result in not enough resources available for the lower priority tasks. As outputs of this model we obtain three different matrices containing the delays, the buffer requirements and the remaining services.

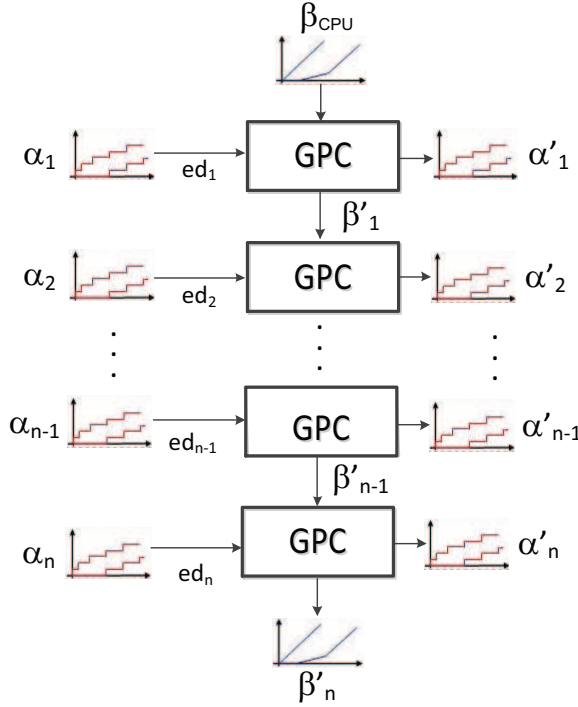


Figure 3.5: Block view of the suggested FP model.

Chapter 4

Case study

In this chapter we apply the proposed design space exploration framework in the previous chapter for two different cases. There are many design spaces, such as timing, resource usage, energy consumption, etc., to be explored. As mentioned in Chapter 1, we want to study the timing quality, in particular the delays of the stream reasoning applications. We have constructed two general performance models (FIFO and FP) in the previous chapter. The analysis is conducted at first over a large range of data with large granularity between each iteration then reduce the resolution down to study important parts with more detailed granularity, if there exists such a range.

4.1 Experimental setup

The actual hardware platform in this case is a Pentium III with a processor speed of 700 MHz. The applications are modeled using DyKnow [2]. There is one stream with logical formula: *Always Not p Implies Eventually Always p*. This stream contains states. In each state a different value is assigned to the variable p in the logical formula. When the state in the formula is updated, we get a new task. These are the tasks that we have defined in the problem formulation in Chapter 1. Updating the formula requires different amount of timing depending on what has happened before. This can be modeled with one task per state of the formula updating [1]. The events are modeled according to the common event pattern of (p,j,d) as mentioned earlier and the default information are given in Tables 4.1 and 4.2.

Based on the default data we roughly estimate the range of task parameters which is of interest to the analysis. The tasks are performed with repeating parameters and all their possible combinations. The combination of the task parameters can be arranged as shown in Fig. 3.3. For both scheduling methods we vary the periods, the jitters and the number of in-

Task	Period	Jitter	WCET
T_1	70	0	0.07
T_2	60	0	0.06
T_3	30	0	0.03

Table 4.1: Default task information for First-in-First-out scheduling.

Task	Period	Jitter	WCET	BCET
T_1	70	0	0.07	0.045
T_2	60	0	0.06	0.03
T_3	30	0	0.06	0.06
T_4	30	0	0.03	0.03

Table 4.2: Default task information for Fixed-Priority scheduling.

stances. When changing the number of instances (the number of instances here means how many copies of a certain tasks that are executed at a certain time), the WCET and BCET are affected (see example 4.1). Since Matlab is ignoring the infinite values, all infinite numbers are automatically converted to a high number. The Matlab version used for this study is R2010b.

Example 4.1

The number of instances means the number of copies of that task. All instances are considered as a single task but the total worst-case execution time is affected. Considering an example task with WCET of 30 ms, we run 1000 instances of that task, then the WCET will be $30 \times 10^{-3} \times 10^3 = 30$ s.

4.2 First in first out analysis

In this section, we present the results after the formal analysis with the FIFO scheduler according to Fig. 3.4. We consider three cases with varying (i) the number of instances, (ii) the periods and (iii) the jitters of all three tasks.

Case 1 - Instance variation

In this case, we vary the number of instances according to Table 4.3. The number of instances is varied between 1 and 50. When varying the number of instances, the granularity is 5. The periods are fixed at the default periods which presented in Table 4.1 and the jitters are fixed at zero for all three tasks.

Task	Period	Jitter	WCET
T_1	70	0	0.07 (1...50)
T_2	60	0	0.06 (1...50)
T_3	30	0	0.03 (1...50)

Table 4.3: Task set when varying the number of instances.

With fixing the periods and jitters, the delays of the three tasks are found to increase linearly with increasing the number of instances.

Case 2 - Period variation

For the first case we vary the periods of all tasks according to Table 4.4. The number of instances is fixed to 10 and the jitter is fixed at zero for all tasks. We guess the range of the periods which might be of interest for this study which is between 5...70 ms. For each analysis, the periods are varied with a granularity of 5 ms.

Task	Period	Jitter	WCET
T_1	5...70	0	0.7
T_2	5...70	0	0.6
T_3	5...70	0	0.3

Table 4.4: Task set when varying the periods.

Case 3 - Jitter variation

For this case we vary the jitters of all the three tasks according to Table 4.5. The range which the jitters was varied is between 0 and 70 ms. The granularity of the jitter variation is 5 ms. We keep the number of instances fixed at 10 like in the case 1 and the periods for the three tasks are fixed at the default periods presented in Table 4.1.

Task	Period	Jitter	WCET
T_1	70	0...70	0.7
T_2	60	0...70	0.6
T_3	30	0...70	0.3

Table 4.5: Task set when varying the jitters.

For both period and jitter variations we can conclude as two different cases: (i) *jitter* < *period* and (ii) *jitter* ≥ *period*.

Case (i): $jitter < period$ (including the case for $jitter = 0$ and the periods varied between 5. . . 70) the delays of all the tasks are always 1.6 ms no matter the change of periods or jitter.

Case (ii): $jitter \geq period$ the delays of all the tasks change to a different value but the delays are still the same for all three tasks (see example 4.2).

From here we can make a conclusion which is only true in this study that FIFO scheduling is not a good option for the DyKnow applications because we can not distinguish between the delays of the tasks. Furthermore we can not know which task have higher priority, since the delays are same for all the tasks.

Example 4.2

Considering two tasks with the following parameters in order to explain why the delay of task 3 did not vary despite the variation of the periods and jitters.

- Task 1: $P_1 = 3$ ms, $J_1 = 0$, minimum inter-arrival distance = 0 and execution demand = 1 ms
- Task 2: $P_2 = 4$ ms, $J_2 = 0$, minimum inter-arrival distance = 0 and execution demand = 2 ms

1. First case: we schedule two tasks using the FIFO component with task 1 arrives first and task 2 arrives just right after. This means the worst case delay for task 2 would be the sum of the execution demands of both task 1 and task 2, which is 3 ms.
2. Second case: now considering the opposite of case 1 i.e., task 2 arrives first. This will also result in the same sum of the delays for the two tasks (3 ms).

4.3 Fixed priority analysis

In this study, we deal with four tasks by applying the Fig. 3.5 proposed in Chapter 4 and using the task information from Table 4.2. We study the influence on the delay of task 3 by varying the parameters of tasks 1 and 2 while keeping the parameters of task 3 fixed. When we monitor the impact on the delay of task 3, the delay of task 4 will be ignored because it is the lowest priority task and does not affect any other tasks. We also consider three cases with varying (i) the periods, (ii) the jitters and (iii) the number of instances for all four tasks.

Case 1 - Period variation

In this case, the periods for all four tasks are varied according to Table 4.6. We simply guess the range of the period which we think is interesting which is between 5 and 70 ms. The same granularity has been used like case 1 with the FIFO scheduler which is an interval of 5 ms between each analysis. The number of instances of all four tasks are fixed at 100 and the jitters of the four tasks are fixed at zero.

Task	Period	Jitter	WCET	BCET
T_1	5...70	0	7	4.5
T_2	5...70	0	6	3
T_3	5...70	0	6	6
T_4	5...70	0	3	3

Table 4.6: Task set when varying the periods.

We are interested in studying two cases: 1) task 3's maximum delay, D_3 with task 3's period fixed at different values and 2) the maximum delays for all four tasks when their periods are set to the same value. For all the cases of period variations, D_3 never exceed 59 ms so therefore after 59 ms we consider the delay is infinite and will be set to 60 ms. D_3 will be represented by both 2D- and 3D-graphs. Our experiments show that with the period of task 3, P_3 , fixed at 5 ms, its delay becomes infinite for all values of P_1 and P_2 in the range 5-70 ms. As can be seen in Fig. 4.1a, if P_1 and P_2 are larger than 30 ms, D_3 will be stable which is the dark-blue area in the Fig. 4.6. When P_1 and P_2 are less than 15 ms, D_3 approaches infinite values, consequently the dark-red region as shown in the Figure 4.6.

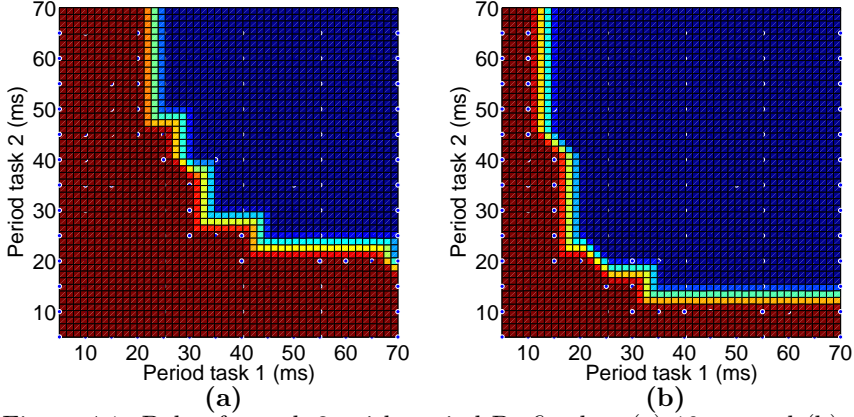


Figure 4.1: Delay for task 3, with period P_3 fixed at (a) 10 ms and (b) 15 ms.

The delay-graphs will be shown for P_3 between 10ms and 55 ms because below 10 ms, D_3 will always be infinite and above 55 ms D_3 will be similar to two previous graphs. From Figures 4.1-4.5, D_3 are represented with 2D graphs (where x-axis is variation of P_1 , y-axis is variation of P_2 and the contour of D_3 depending on P_1 and P_2) with three different regions where P_3 is fixed at different values. The dark-red area is where D_3 is infinite, the dark-blue area is where D_3 is stable and the line that clearly distinguish these two regions is where D_3 may potentially get infinite.

Fig. 4.1a shows D_3 with its period being fixed at 10 ms while P_1 and P_2 vary in the range of 5-70 ms. This also means that we can have many different variation of P_1 and P_2 , but there are trade-offs. Comparing the case when P_1 is 25 ms and P_2 is 50 ms and the case when P_1 is 35 ms and P_2 is 30 ms, the total period of three tasks is 85 ms and 75 ms, respectively. We notice here that by the later choice we can already save 10 ms of resource usage. When P_3 increases from 10 ms to 15 ms the stable region of D_3 also increases which can be seen in Fig. 4.1b. Here we also have many possibilities of the choices of P_1 and P_2 . Consider the comparison of the case when P_1 is 15 ms and P_2 is 45 ms with the case when P_1 is 25 ms and P_2 is 20 ms, the total period of three tasks here is 75 ms and 60 ms, respectively. Here we spared 5 ms more compared with the case when P_3 is 10 ms and D_3 is still in the stable region. If recall from Chapter 3 at requirement 2a, when varying the periods the following condition $D_i \leq P_i$ should hold. In this case the dark-blue region depicts D_3 stable at 19 ms which is larger than P_3 .

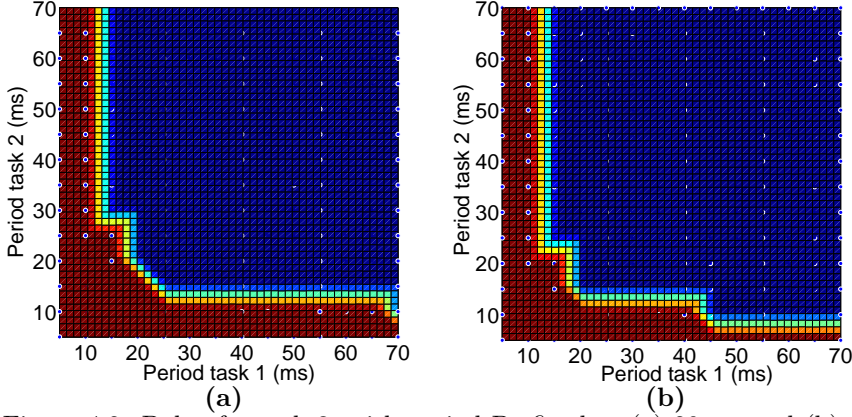


Figure 4.2: Delay for task 3, with period P_3 fixed at (a) 20 ms and (b) 25 ms.

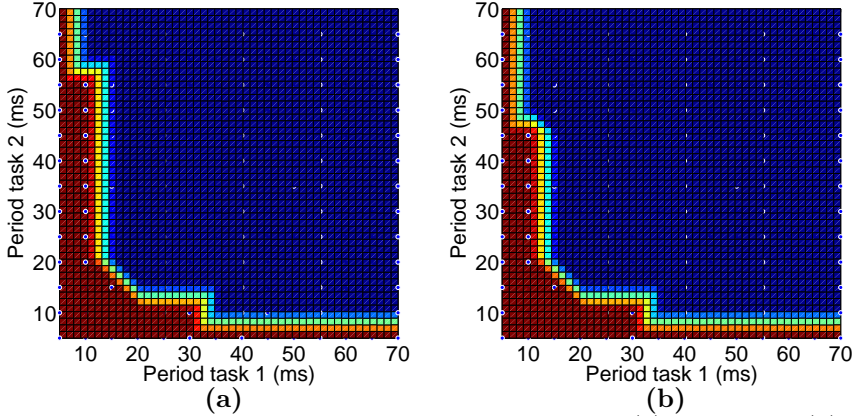


Figure 4.3: Delay for task 3, with period P_3 fixed at (a) 30 ms and (b) 35 ms.

With P_1 , P_2 varied between 5-70 ms and P_3 is fixed at different values between 20-35 ms, the stable region for D_3 does not changes much compared to when P_3 is fixed at 15 ms, which can be seen in Figures 4.2-4.3. As shown in Fig. 4.2a, if P_1 and P_2 is selected as 25 ms and 15 ms, respectively, P_3 is fixed at 20 ms. The total period of all three tasks is 60 ms and D_3 is 19 ms which is smaller than P_3 . Consider the case when P_3 is fixed at 25, 30 and 35 ms with P_1 and P_2 is selected to 20 ms and 15 ms, respectively, D_3 behaves the same way as the case when P_3 is fixed at 20 ms.

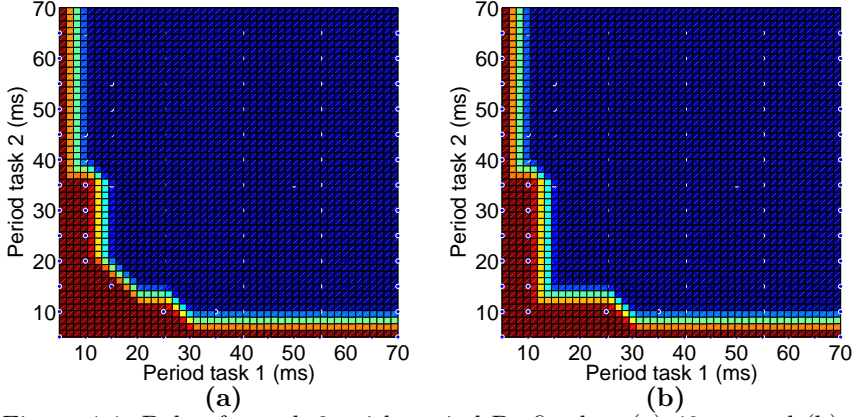


Figure 4.4: Delay for task 3, with period P_3 fixed at (a) 40 ms and (b) 45 ms.

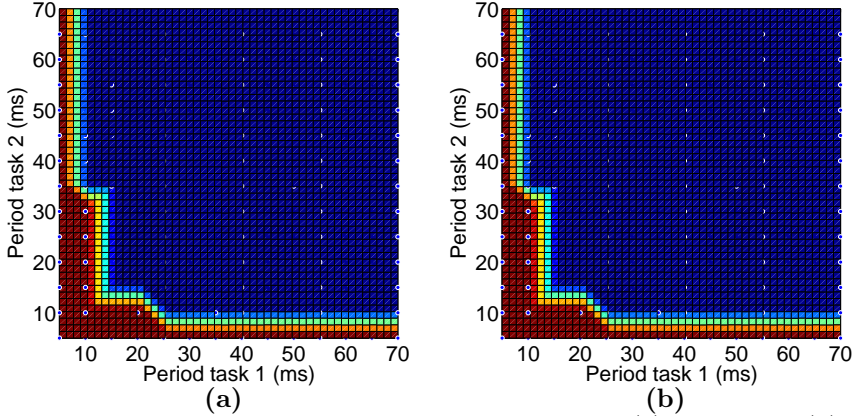


Figure 4.5: Delay for task 3, with period P_3 fixed at (a) 50 ms and (b) 55 ms.

With P_1 , P_2 varied between 5-70 ms and P_3 is fixed at different values between 40-55 ms, the stable region for D_3 does not changed much compared to when P_3 is fixed at 35 ms, which can be seen in Figures 4.4-4.5. As shown in Fig. 4.4a, if P_1 and P_2 is selected as 25 ms and 15 ms, respectively, P_3 is fixed at 40 ms. The total period of all three tasks is 80 ms and D_3 is 19 ms which is smaller than P_3 . Consider the case for when P_3 is fixed at 45, 50 and 55 ms with P_1 and P_2 are selected to 20 ms and 15 ms, respectively, D_3 behaves the same way as the case when P_3 is fixed at 40 ms.

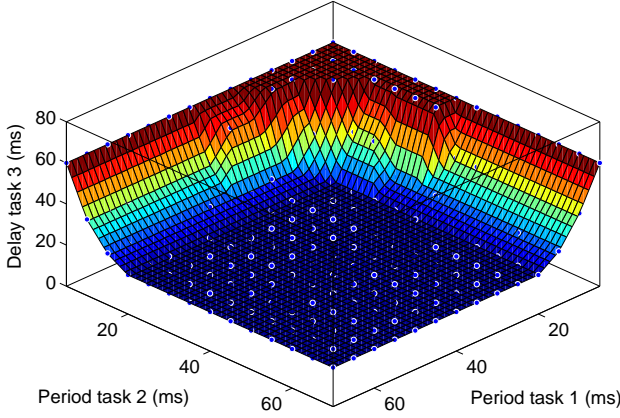


Figure 4.6: Delay for task 3, with P_3 period fixed at 40 ms.

So far we have only displayed 2D-graphs of D_3 . Fig. 4.6 shows D_3 when fixing P_3 at 40 ms while P_1 and P_2 are varied between 5-70 ms. The reason we only show a 3-D graph of D_3 with P_3 fixed at 40 ms is because we choose this period to be the default one. From the designer's point of view this 3D graph is not as straight forward as the 2D graphs. We can see that on the border between the stable delay region and the infinite delay region there are observation points that is interesting to study which can be carried out by varying a more detailed granularity around that region.

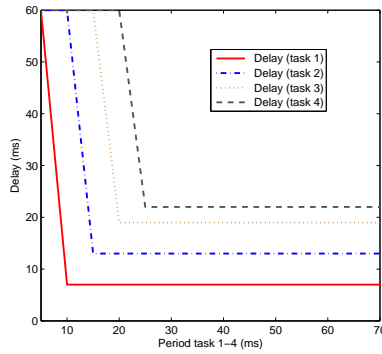


Figure 4.7: Delay of four tasks with period variation in the range 5-70 ms.

Now we consider the influence of the periods P_1 - P_4 on the delay of each task. The x-axis in Fig. 4.7 shows when the periods of all four tasks are set at the same value while the y-axis shows the delay of each task. The

periods are varied in the range 5-70 ms. As can be seen in the figure, if the period of all four tasks is smaller than 10 ms the resulting delay of T_{2-4} already becomes infinite and hence the system will not work. Reasoning of two tasks requires the period of task 1 and task 2 to be at least 12 ms or larger $P_{1,2} \geq 12$ ms. For the case with three tasks, as already discussed previously, a finite delay of task 3 is ensured if the period of all three tasks are larger than 20 ms, $P_{1,2,3} \geq 20$ ms. As shown in the figure, a finite delay for all four tasks can be guaranteed if the periods of all four tasks is in the range $25 \leq P_{1,2,3,4} \leq 70$ ms. Thus, in the case of four tasks, choosing all the periods $P_{1,2,3,4}$ of 25 ms can guarantee minimal resource usage.

Case 2 - Jitter Variation

In this case, we vary the jitter of task 1, J_1 and task 2, J_2 , and monitor how their variation affects the delay on task 3. We fix the number of instances of all the three tasks at 100 and the periods at zero. Since the default jitters for all the three tasks are zero, the lower limit of the jitters should be zero. The upper limit is 70 ms, because 70 ms is the longest period out of the three. In this case, when we are studying the delay of task 3 then we are ignoring task 4 because it does not have any affects on the higher priority tasks.

Task	Period	Jitter	WCET	BCET
T_1	40	0...70	7	4.5
T_2	40	0...70	6	3
T_3	40	0...70	6	6
T_4	40	0...70	3	3

Table 4.7: Task set when varying the jitters.

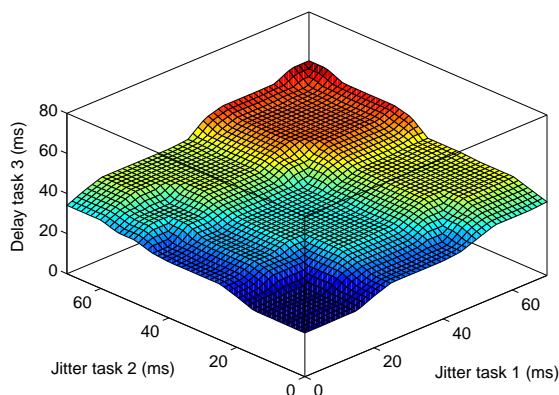


Figure 4.8: Delay for task 3, with jitter fixed at 10 ms in 3D-view.

In the case of jitter for task 3 being fixed at 5 ms, the delay for task 3, D_3 varies as shown in Figures 4.8. D_3 first increases and then stays constant for an amount of time which can be seen as different plateaus in the figure. D_3 will keep increasing if J_1 and J_2 increases. The reason for why D_3 is increasing as different plateaus is because, if we allow more jitter of higher priority tasks, we allow more higher priority tasks to arrive and therefore the lower priority task (in this case T_3) has to wait till both T_1 and T_2 finishes running. In some range where we vary the jitter, it does not affect more instances of higher priority tasks to arrive and therefore we can see a

constant plateau.

Figure 4.9 shows the delay for all three tasks with the same jitter. The x-axis in the figure represents the jitters of all four tasks are set at the same value while the y-axis shows the delay of each task. As shown in the figure, the delays increase accordingly with the variation of the jitters. If we set the same jitter of all the three tasks and vary them with the same amount, the delays of all the four tasks change similarly to the case when fixing the jitter of task 3 (in Fig. 4.8)

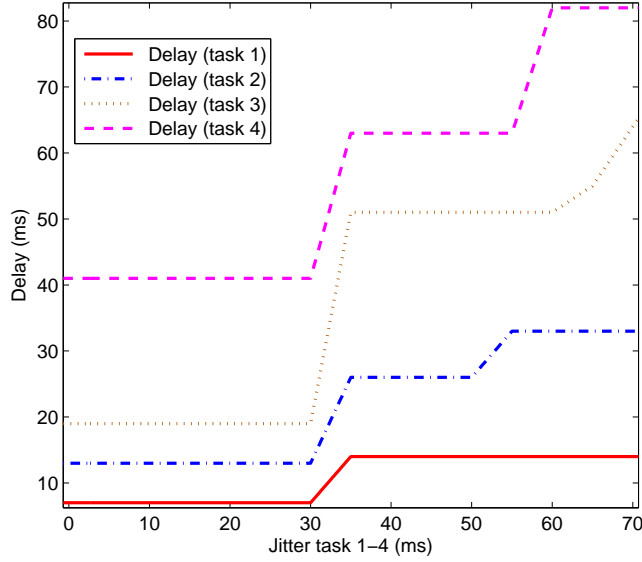


Figure 4.9: Delay of four tasks with jitter variation in the range 0-70 ms.

Throughout the variations of J_1 and J_2 , D_3 behaves in similar pattern. D_3 is raised upwards in the Z-direction.

Case 3 - Instance variation

In this case, we study the change of the delays with respect to the variation of the number of instances with fixed periods and jitters of all the three tasks at 40 ms and zero, respectively. The minimum number of instances for each task is set to 1 while the maximum number of instances can be estimated from the WCETs of all four tasks ($0.045+0.03+0.06+0.03= 0.165$ ms) and their fixed period (40 ms). The maximum number of instances is estimated to be $40/0.165 = 242$ (~ 300) and is rounded to the nearest hundred. The granularity of the number of instances variation is 20.

Task	Period	Jitter	WCET	BCET
T_1	40	0	0.07 (1...300)	0.045 (1...300)
T_2	40	0	0.06 (1...300)	0.03 (1...300)
T_3	40	0	0.06 (1...300)	0.06 (1...300)
T_4	40	0	0.03 (1...300)	0.03 (1...300)

Table 4.8: Task set when varying the number of instances.

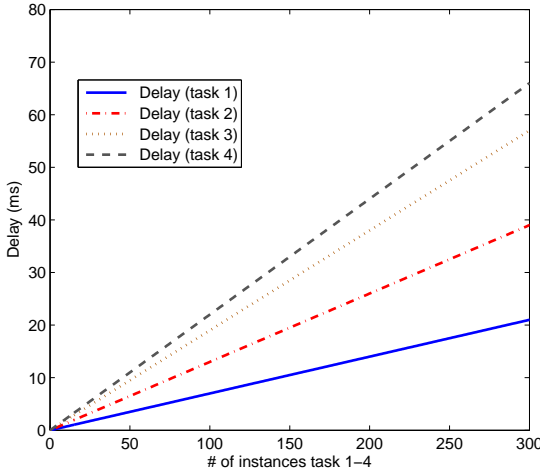


Figure 4.10: Delay of four tasks with instance variation in the range 1-300.

The result of the experiment is shown in Figures 4.10 and 4.11. We notice that the delays increase linearly with increasing the number of instances as can be seen in Fig. 4.10. The x-axis in the figure shows the number of instances for all four tasks set at the same value while the y-axis is the delay of each task.

The delay of task 3, D_3 , fixed at 100 instances and the number of instances of task 1 and task 2 varying between 1 to 300. As can be seen in Fig. 4.11, that D_3 increases linearly with increasing of the number of instances of tasks 1 and 2 up to 300.

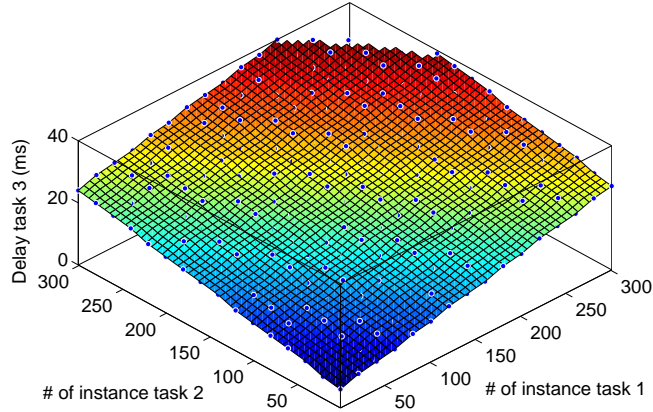


Figure 4.11: Delay of task 3 with fixed number of instances in 3D view.

Chapter 5

Conclusion

In summary, we have used RTC for the design space exploration of the QoS for stream reasoning applications on board a UAV using an abstract performance model of the system. The dependences of the delays was studied on

- the variation of the periods of tasks with fixed jitters and fixed number of instances,
- the variation of the jitters with fixed periods of tasks and fixed number of instances, and
- the number of instances with fixed periods and jitters

We applied the proposed design space exploration framework to two specific cases, FIFO and FP scheduling. For each case there are three different design space exploration options.

For FIFO scheduling we found that the delays of the stream reasoning applications did not vary despite the fact that the application parameters (jitters and periods) varied. The reason is that the function implemented (`rtcfifo`) in RTC toolbox is dealing with the worst-case behaviors and therefore the delays of each tasks is the total execution time of all the tasks. When varying the number of instances the delays are increasing linearly.

For FP scheduling the most interesting part of the design space exploration is when we vary the periods of these applications. There were many possible options of the periods to choose and if we choose the periods of all four tasks, $P_{1,2,3,4} = 25$ ms then the resource usage is optimal. When looking at the 3-D graphs (period variations) of the delays, we also notice that there are observation points on the border between stable and unstable delay. The study can be focused even more with more detailed granularity within this region. The delays of all four tasks are found to be different

plateaus staying constant for a amount of time and then increasing again when varying the jitters. When varying the number of instances the delays give rise to the same pattern as with a FIFO scheduler.

Future work

There are often challenges. The time it takes to get the results after each experiment varies depending on which parameter we vary. For the case when we vary the jitters and number of instances, the time is estimated to roughly eight hours until all experiment finishes. Studying the impact of the periods on the delays takes roughly one to one and a half week depending on the granularity of the study.


In this thesis only the case of a uniprocessor environment was studied. The analysis can be extended to a multiprocessor environment. When dealing with multiprocessor environment we can, for instance, estimate the end-to-end delay of the task after being processed by different resources.

Three different matrices: the delay, buffer and utilization matrices can be obtained from the RTC performance model which has been generated for the analysis. We have done studies for the delays in this thesis. The buffer requirements and CPU utilizations can also be studied in the same approach as used for the delay.

The framework can be improved even further, for example, the different constants declared at the beginning of the main code can be acquired as input from the user instead. This will be more flexible and you do not need to recompile the code when you make changes. As a start of this framework, it is developed as a terminal based program. This can be improved by making a graphical user interface.

Bibliography

- [1] F. Heintz, *DyKnow: A Stream-Based Knowledge Processing Middleware Framework*. PhD thesis, 2009.
- [2] F. Heintz, J. Kvarnström, and P. Doherty, “Bridging the sense-reasoning gap: DyKnow - Stream-based middleware for knowledge processing,” *Advanced Engineering Informatics*, vol. 24, no. 1, pp. 14–26, 2010.
- [3] S. Chakraborty, S. Künzli, and L. Thiele, “A general framework for analysing system properties in platform-based embedded system designs,” in *Design Automation and Test in Europe (DATE)*, (Munich, Germany), pp. 190–195, IEEE Press, 2003.
- [4] E. Wandeler and L. Thiele, “Real-Time Calculus (RTC) Toolbox.” <http://www.mpa.ethz.ch/Rtctoolbox>.
- [5] E. Wandeler and L. Thiele, “Abstracting functionality for modular performance analysis of hard real-time systems,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, (Shanghai, P.R. China), pp. 697—702, 2005.
- [6] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, “System architecture evaluation using modular performance analysis - a case study,” *Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 649 – 667, 2006.
- [7] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis - the symta/s approach,” in *IEEE Proceedings Computers and Digital Techniques*, 2005.
- [8] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Berlin, Heidelberg: Springer-Verlag, 2001.
- [9] W. Haid and L. Thiele, “Complex task activation schemes in system level performance analysis,” in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis, CODES+ISSS '07*, (New York, NY, USA), pp. 173–178, ACM, 2007.

 Linköpings universitet	Avdelning, Institution Division, Department ESLAB, Software and Systems, Department of Computer and Information Science SE-581 83 Linköping, Sweden	Datum Date August 9, 2012
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> övrig rapport <input type="checkbox"/> _____	ISBN - <hr/> ISRN LIU-IDA/LITH-EX-A-12/027-SE <hr/> Serietitel och serienummer ISSN Title of series, numbering -
URL för elektronisk version http://www.ep.liu.se		
Titel Title Design Space Exploration of the Quality of Service for Stream Reasoning Applications Författare Author Viet Ha Nguyen		
Sammanfattning Abstract <p>An Unmanned Aerial Vehicle (UAV) is often an aircraft with no crew that can fly independently by a preprogrammed plan, or by remote control. Several UAV applications, like autonomously surveillance and traffic monitoring, are real-time applications. Hence tasks in these applications must complete within specified deadlines.</p> <p>Real Time Calculus (RTC) is a formal framework for reasoning about real-time systems and in particular streaming applications. RTC has its mathematical roots in Network Calculus. It supports timing analysis, estimating loads and predicting memory requirements.</p> <p>In this thesis, a formal analysis of real-time stream reasoning for UAV applications is conducted. The performance analysis is based on RTC using an abstract performance model of the streaming reasoning on board a UAV. In this study, we consider two different scheduling methods, first-in-first-out (FIFO) and fixed priority (FP). In the FIFO scheduling model the priorities of the tasks are assigned and processed based on the order of their arrival, while in the FP scheduling model the priorities of the tasks are preassigned. The Quality of Service (QoS) of these applications is calculated and analyzed in a proposed design space exploration framework.</p> <p>QoS can be defined differently depending on what field we are studying and in this thesis we are interested in studying the delays of the real-time stream reasoning applications when (i) we fix jitters and number of instances and vary the periods, (ii) we fix the periods and number of instances and vary the jitters, and (iii) we fix the periods, jitters and vary the number of instances.</p>		
Nyckelord Keywords real time calculus, RTC, quality of service, QoS, delays, design space exploration		