

Automatic Synchronisation Detection in Petri Net Performance Models Derived from Location Tracking Data

Nikolas Anastasiou¹, William Knottenbelt¹, and Andrea Marin²

¹Department of Computing, Imperial College London
South Kensington Campus, London SW7 2AZ
{na405,wjk}@doc.ic.ac.uk

²D.A.I.S., Università Ca' Foscari Venezia
via Torino, 155, Venice, Italy
marin@dsi.unive.it

Abstract. The inference of performance models from low-level location tracking traces provides a means to gain high-level insight into customer and/or resource flow in complex systems. In this context our earlier work presented a methodology for automatically constructing Petri Net performance models from location tracking data. However, the capturing of synchronisation between service centres – the natural expression of which is one of the most fundamental advantages of Petri nets as a modelling formalism – was not explicitly supported. In this paper, we introduce mechanisms for automatically detecting and incorporating synchronisation into our existing methodology. We present a case study based on synthetic location tracking data where the derived synchronisation detection mechanism is applied.

Keywords: Location Tracking, Performance Modelling, Data Mining, Generalised Stochastic Petri Nets

1 Introduction

The proliferation of GPS-enabled mobile devices, RFID tags and high-precision indoor location tracking systems has led to the widespread availability of detailed low-level data describing the movements of customers and/or resources. One way to practically exploit this data in order to gain insight into high-level system operation is to automatically derive a performance model in the form of a queueing network [8] or Stochastic Petri Net [1].

Such models enable us to extract useful information about customer and resource flow and to identify possible bottlenecks in the underlying system. Once a realistic and accurate model has been constructed it can be also used as a predictive tool; for example, the model can be modified to examine the system's performance under hypothetical scenarios, i.e. addition/removal of resources or increased workload.

Our earlier work [1] has presented a methodology which is able to automatically construct Generalised Stochastic Petri Net [2, 9] performance models from raw location tracking data. However, synchronisation between service centres was not explicitly captured. This is a serious deficiency since many physical customer processing systems such as hospitals, airports and car assembly lines exhibit many instances of synchronisation. For example, if we consider a treatment room in a hospital, the examination of a patient cannot occur without the presence of a doctor. Thus, the purpose of the present paper is to introduce a mechanism for automatically detecting and incorporating synchronisation into our existing methodology.

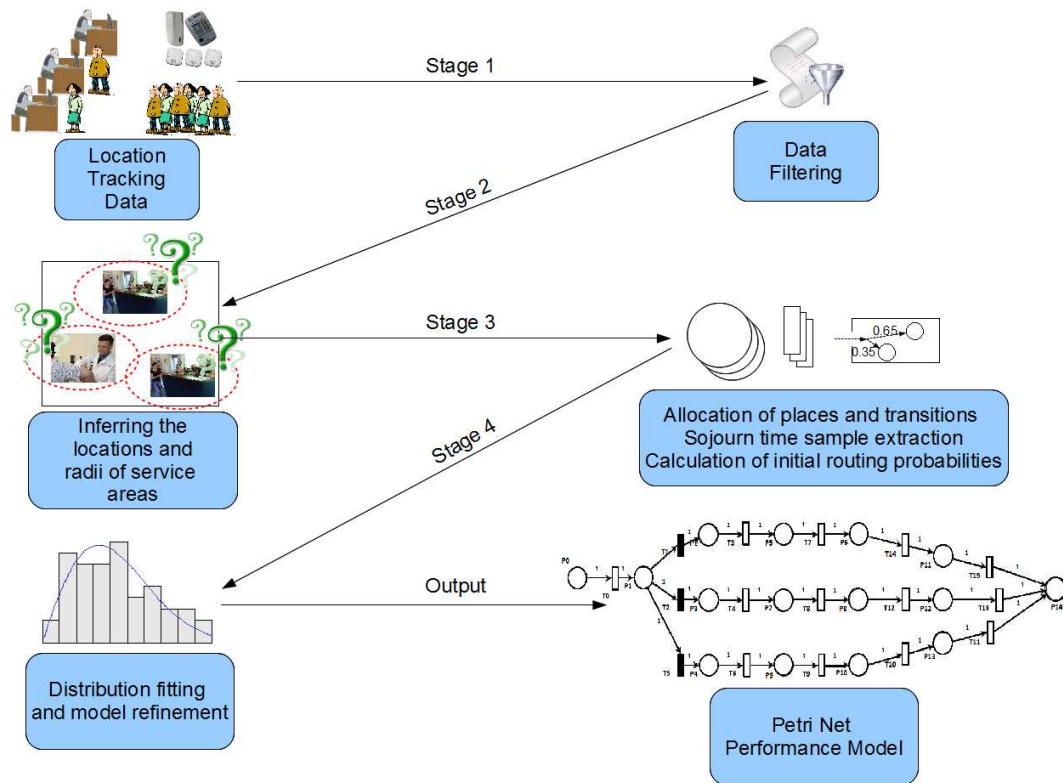


Fig. 1. The four-stage data processing pipeline as in [1].

Our existing methodology is based on a four-stage data processing pipeline (see Figure 1). The first stage of the pipeline performs some basic data filtering. The second stage is responsible for inferring the locations and radii of service areas in the system. The third stage constructs a set of places and transitions to represent customer flow in the system. Sojourn time samples for each customer

processed within each service area are also extracted, and separated into waiting time and service time. Likewise, travelling time samples between pairs of service areas are extracted. Finally, a simple counting mechanism is used to calculate the initial routing probabilities of the customers in the system. The fourth stage uses the G-FIT tool [10] to fit a Hyper-Erlang distribution (HErD) [6] to the extracted service time and travelling time samples and refines both the structure and parameters of the model accordingly. The inferred model is stored in the portable PNML format [3] and can be visualised using PIPE2, an open source platform independent Petri Net editor[4], amongst other tools.

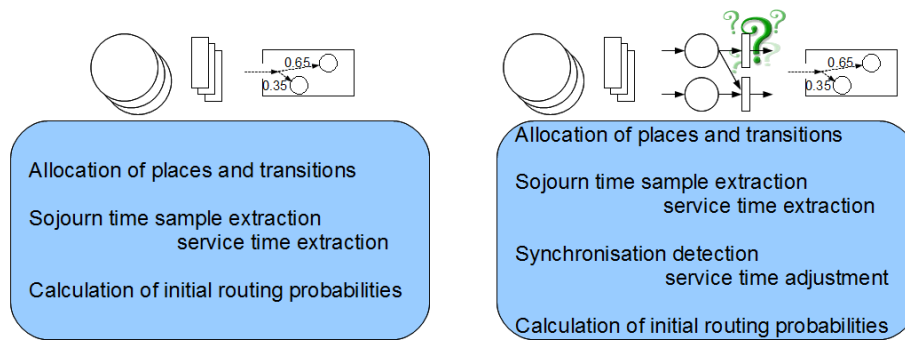


Fig. 2. A high-level description of the third stage of the processing pipeline before (left) and after (right) the incorporation of the synchronisation detection mechanism.

Here we focus on the third stage of the data processing pipeline. As shown in overview in Figure 2, and as described in detail in Section 2, we expand this stage by specifying an additional task which detects synchronisation between service areas. Section 3 presents a case study based on synthetic location tracking data generated using an extended version of LocTrackJINQS [7]. Section 4 concludes and considers future work.

2 Synchronisation Detection Mechanism

Our aim is to construct a conservative scheme to determine whether the processing of customers at each service area is subject to some synchronisation conditions (expressed as conditions on the number of customers present within other service areas) at certain time points. To this end, we have designed three functions (see Algorithms 1, 2 and 3) that can be applied together to perform the synchronisation detection task.

To formalise our approach, we introduce some notation:

- N is the total number of service areas inferred from the second stage of the processing pipeline,

- $P = \{P_1, P_2, \dots, P_N\}$ is the set of inferred service areas (subsequently represented by places in the derived Petri net model),
- $C_i = \{c_i^{(1)}, c_i^{(2)}, \dots, c_i^{(n_i)}\}$ is the multiset¹ of all customers processed by P_i , with $n_i = |C_i|$,
- $e_i^{(j)}$ is the timestamp of the entry of $c_i^{(j)}$ into P_i ,
- $s_i^{(j)}$ is the service initiation timestamp of $c_i^{(j)}$, and
- $f_i^{(j)}$ is the service termination timestamp of $c_i^{(j)}$.
- $M_i(t)$ is the number of customers present on service area P_i at timestamp t . This is also referred to as the *marking* of P_i at time t .
- $M_i(t_1, t_2)$ is the maximum number of customers observed on service area P_i during the time interval $[t_1, t_2]$. This is also referred to as the *maximum marking* of P_i during the time interval $[t_1, t_2]$.

Given a service area P_i , we consider the processing of each customer that receives service there in turn. Our approach is based on finding evidence – for each customer – that its processing may have been dependent on the presence of customers on other service areas. This evidence has two components: one is the maximum marking observed on each of the other service areas during the interval during which the customer was serviced; the other is the marking observed on each of the other service areas at the instant of termination of the customer’s service. These two components are combined across all customers processed by P_i – taking into account of the possibility of error and noise – to yield the likely synchronisation conditions of service at P_i ². Formally:

Definition 1. *The j th customer $c_i^{(j)} \in C_i$ is said to receive service at P_i with possible synchronisation from each service area $P_k, k = 1, \dots, N$ and $k \neq i$, if:*

$$M_k(s_i^{(j)}, f_i^{(j)}) > 0, \quad (1)$$

and

$$M_k(f_i^{(j)}) > 0 \quad (2)$$

Definition 2. *Synchronisation between server P_i and server(s) $P_k, k = 1, \dots, N$ and $k \neq i$, is inferred if the synchronisation percentage s_p defined as,*

$$s_p(i, k) = \frac{|\{c_i^{(j)} \mid M_k(s_i^{(j)}, f_i^{(j)}) > 0, M_k(f_i^{(j)}) > 0, j = 1, \dots, n_i\}|}{n_i} \quad (3)$$

satisfies

$$s_p \geq s_{thresh} \quad (4)$$

where s_{thresh} is the hypothesis acceptance threshold.

The value of the acceptance threshold (typically in the range $[0.8, 1]$) can be chosen according to factors such as the precision of the location tracking system used, tag update rate and topology of the system being modelled.

¹ Thus supporting the possibility of multiple service periods for the same customer.

² Here we assume a single class of customers. It is straightforward to apply the combination across each customer class in a scenario with multiple customer classes.

2.1 Algorithm Description

Algorithm 1 and Algorithm 2 present auxiliary functions that straightforwardly compute $M_k(t)$ and $M_k(t_s, t_f)$ respectively.

The main function, `computeSynchronisation` (see Algorithm 3), is applied in turn to every service area $P_i \in P$. There are two phases in this algorithm. In the first phase (see lines 4 to 12) we construct the `csmMatrix`, which describes the possible set of synchronisation dependencies between the service of customer j at P_i , and the markings of the other service areas. In this phase, for each P_k , $i \neq k$, we also count the number of customers for which a potential synchronisation was observed, during their service at P_i . In the second phase (see lines 13 to 18) we calculate $s_p(i, k)$ and if its value exceeds the value of s_{thresh} we compute the synchronisation marking on P_k required to support service at P_i . This is computed as a low percentile of the set of synchronisation markings; this is preferred to simply taking the minimum because it is more robust to measurement errors inherent in location tracking systems. This percentile is determined by the function `percentile(M, α)` (see line 16, Algorithm 3) which computes the α th percentile of the set M .

Algorithm 1 : $M(k, t) : \text{int}$

```

1: marking  $\leftarrow$  0
2: for  $j = 1$  to  $n_k$  do
3:   if  $c_k^{(j)}$  was present in  $P_k$  at  $t$  then
4:     marking  $\leftarrow$  marking + 1
5:   end if
6: end for
7: return marking

```

Algorithm 2 : $M(k, t_s, t_f) : \text{int}$

```

1: maxMarking  $\leftarrow$   $M(k, t_s)$ 
2: for  $j = 1$  to  $n_k$  do
3:   if  $t_s \leq e_k^{(j)} < t_f$  then
4:     instMarking  $\leftarrow$   $M(k, e_k^{(j)})$ 
5:     if instMarking  $>$  maxMarking then
6:       maxMarking  $\leftarrow$  instMarking
7:     end if
8:   end if
9: end for
10: return maxMarking

```

If we assume that $\forall i, n_i = n$, then the worst-case time complexity of the function `computeSynchronisation` and synchronisation detection for the entire

Algorithm 3 : `computeSynchronisation(i, s_{thresh}) : int[N]`

```

1: customersWithSynch ← new int[ $N$ ] = {0,0,...,0}
2: synchMarking ← new int[ $N$ ] = {0,0,...,0}
3: csmMatrix ← new int[ $n_i$ ][ $N$ ] = { {0,0,...,0}, {0,0,...,0}, ...,
  {0,0,...,0} }
4: for  $j = 1$  to  $n_i$  do
5:   for  $k = 1$  to  $N$  do
6:     if  $k == i$  then continue
7:     csmMatrix[ $j$ ][ $k$ ] ← min( $M(k, s_i^{(j)}, f_i^{(j)})$ ,  $M(k, f_i^{(j)})$ )
8:     if csmMatrix[ $j$ ][ $k$ ] > 0 then
9:       customersWithSynch[ $k$ ] ← customersWithSynch[ $k$ ] + 1
10:    end if
11:   end for
12: end for
13: for  $k = 1$  to  $N$  do
14:   if  $k == i$  then continue
15:   if customersWithSynch[ $k$ ] /  $n_i \geq s_{thresh}$  then
16:     synchMarking[ $k$ ] ← percentile({csmMatrix[1][ $k$ ], ..., csmMatrix[ $n_i$ ][ $k$ ]}, 5)
17:   end if
18: end for
19: return synchMarking

```

network are $O(N \cdot n^3)$ and $O(N^2 \cdot n^3)$ respectively. Based on the same assumption, space complexity is bounded above by the size of `csmMatrix` (see Algorithm 3) and is $O(N \cdot n)$.

Whenever synchronisation is detected involving the processing of customers at P_i , the corresponding service time samples of those customers need to be adjusted to take into account the proportion of time during which the synchronisation condition(s) are satisfied. This is because we assume that service only progresses when the synchronisation condition(s) are met.

2.2 Synchronisation Representation in our Models

After synchronisation between service areas is detected, it needs to be incorporated into the GSPN performance model that is constructed during stage four of the data processing pipeline.

Considering the place representing P_i and its outgoing service transition t_i , then for every place representing P_k such that `synchMarking[k] > 0`, we connect P_k to t_i using a double-headed arc between P_k and t_i with weight equal to `synchMarking[k]`. We use this representation since we are dealing with location tracking environments where customer entities are preserved.

In [1] we described how we fit a HERD to the extracted service time samples of each service area. We represent each HERD by substituting each transition created in the first task of the third stage by a GSPN subnet, as shown in Figure 3. Now this representation is slightly altered to incorporate synchronisation, when detected.

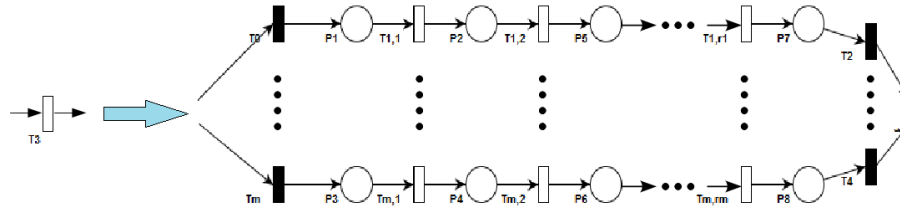


Fig. 3. Replacement of a transition by a GSPN subnet reflecting the fitted HErD (general form).

Let us consider just the transition T_3 in Figure 4 to demonstrate how the synchronisation between P_2 and P_1 is constructed.

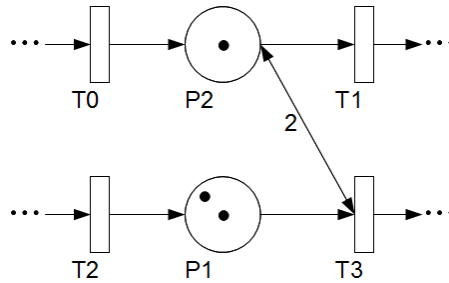


Fig. 4. Modelling synchronisation between server places P_1 and P_2 with P_2 being the synchronising place with a synchronisation marking equal to two.

In order to preserve the synchronisation condition on T_3 as shown in Figure 4, we need to connect P_2 to every immediate transition on the left-hand side of the subnet with arcs – one for each immediate transition – of weight two. Similarly, in order to preserve the number of tokens in P_2 we need to connect every immediate transition on the right-hand side of the subnet to P_2 with arcs of weight two. Assuming that T_3 was replaced by a subnet reflecting a four-state HErD with two Erlang branches, each with two states, the resulting model is shown in Figure 5.

3 Case Study

In this section we conduct a case study to test and demonstrate the developed synchronisation detection mechanism. For this case study we have generated location tracking data using an extended version of LocTrackJINQS [7], which supports synchronisation between pairs of service areas.

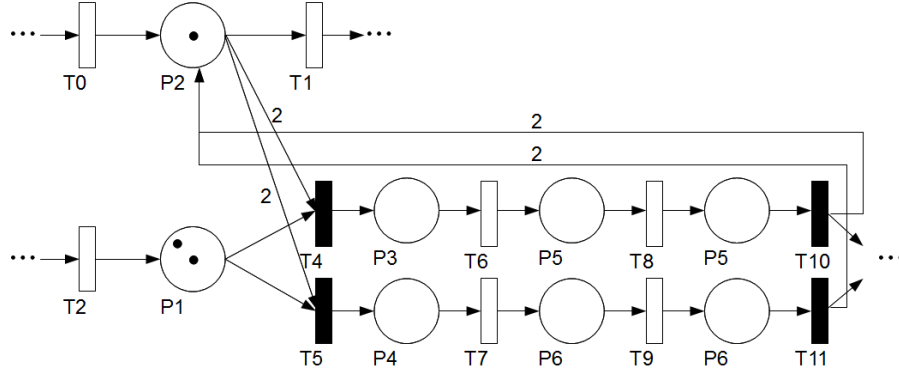


Fig. 5. Modelling synchronisation between places P_1 and P_2 , using a GSPN subnet. Service initiates at P_1 only if P_2 is marked with at least two tokens and P_1 with one.

We present the results for the server location and service radii inference, the synchronisation detection mechanism and the service time distribution fitting (adjusted for synchronisation). Figure 6 shows the experimental setup for the case study and the flow of customers in the system (indicated by arrows). The simulation takes place in a virtual $25\text{m} \times 25\text{m}$ environment and the customers are assumed to travel within the system with speed drawn from a normal distribution with mean 0.5 m/s and standard deviation 0.15 m/s . The location update error, which emulates the standard error of a real-life location tracking system, is also normally distributed with mean 0.15m and standard deviation 0.2m .

	Server Location	Service Radius	Service Time Density
S1	(8.0,5.0)	0.5	Erlang(2, 0.1)
S2	(8.0,15.0)	0.7	Exp(0.1)
S3	(16.0,5.0)	0.6	Erlang(4, 0.3)
S4	(16.0,15.0)	0.5	HErD(2, 3; 0.5, 0.5; 0.08, 0.12)

Table 1. The parameters for each service area in the system, for this case study. The parameters of the HErD represent the phase lengths, weights and rates for each branch respectively, separated by a semi-colon.

Each service area consists of a single customer processing server and has a random customer service discipline. Service areas S_2 and S_3 require at least one customer to be present in service areas S_1 and S_4 respectively in order

to service their customers. The service time for each server follows a different density function. The actual location and service radius of each service area as well as its service time density can be seen in Table 1.

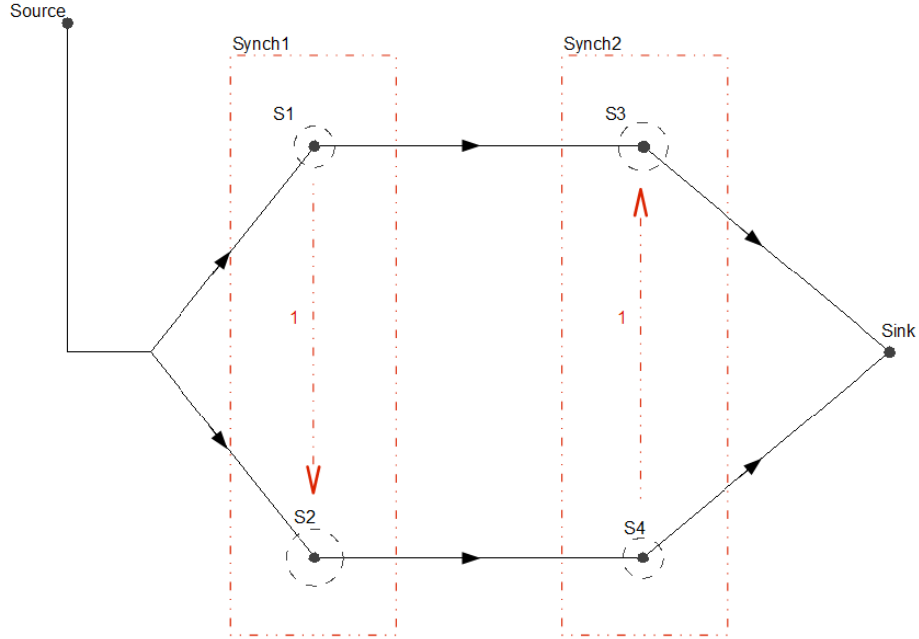


Fig. 6. The experimental setup in terms of abstract system structure. The arrows represent the customer flow in the system and the branching to $S1$ and $S2$ occurs with equal probabilities. The service areas contained in the dotted red rectangles Synch1 and Synch2 are subject to synchronisation. The synchronisation condition is represented by the dotted red arrow. Its source indicates the synchronising service area and its target the service area to be synchronised. The number of customers required to be present in the synchronising service area so that service can be supported in the synchronised service area is denoted by the weight of the dotted red arrow.

3.1 Results

The inferred locations and service radii of the service areas as well as the error between these and their actual values, are depicted in Table 2. From these results we can see that the location and radii of the service areas are approximated very well. The maximum error for the location inference is 0.214 metres and for the service radius approximation is 0.175 metres.

For the evaluation of the sample extraction and HErD fitting process we conduct a Kolmogorov–Smirnov test, to examine the compatibility of the extracted

	Server Location			Service Radius		
	Real	Inferred	Error	Real	Inferred	Absolute Error
S1	(8.0,5.0)	(7.856,4.989)	0.144	0.5	0.566	0.066
S2	(8.0,15.0)	(7.963,14.947)	0.199	0.7	0.839	0.139
S3	(16.0,5.0)	(16.124,4.964)	0.129	0.6	0.759	0.159
S4	(16.0,15.0)	(16.021,14.961)	0.214	0.5	0.675	0.175

Table 2. The inferred location and service radius for each server in the system accompanied with the absolute error, for each case study.

	Service Time Density	Fitted HErD Parameters		
		Phase Lengths	Rate (3 d.p.)	Weights (3 d.p.)
S1	Erlang(2, 0.1)	4	0.222	1.0
S2	Exp(0.1)	1	0.118	1.0
S3	Erlang(4, 0.3)	2, 4, 4	0.052,0.311,23.232	0.153,0.780,0.067
S4	HErD(2,3;0.5,0.5;0.08,0.12)	3	0.136	1.0

Table 3. The parameters of the HErD fitted for each server’s service time density. The parameters of the HErD represent the phase lengths, weights and rate for each branch respectively, separated by a semi-colon.

service time samples for each service area with its best-fit HErD (see Table 4). Similarly to [1], we enumerate all possible HErDs up to a maximum number of states. This number is set equal to ten, i.e. $N = 10$, for all cases where the coefficient of variation of the extracted sample is greater than 0.4 and twenty five, i.e. $N = 25$, when it is less. The best-fit HErD is chosen using the Akaike Information Criterion (AIC) [5].

Figure 8 shows the constructed GSPN performance model in compact transition form. We observe that the structure of the inferred model matches the structure of the abstract simulated system. The transitions between pairs of server places (places that correspond to the service areas) represent the travelling time for each particular pair. The weights of the immediate transitions T_1 and T_0 are 0.457 and 0.543 respectively, approximately matching the simulated routing probabilities of the customer flow which are 0.5 and 0.5. In the model we can also see the constructed synchronisation between S2 and S1 (synchronising service area) as well as between S3 and S4 (synchronising service area).

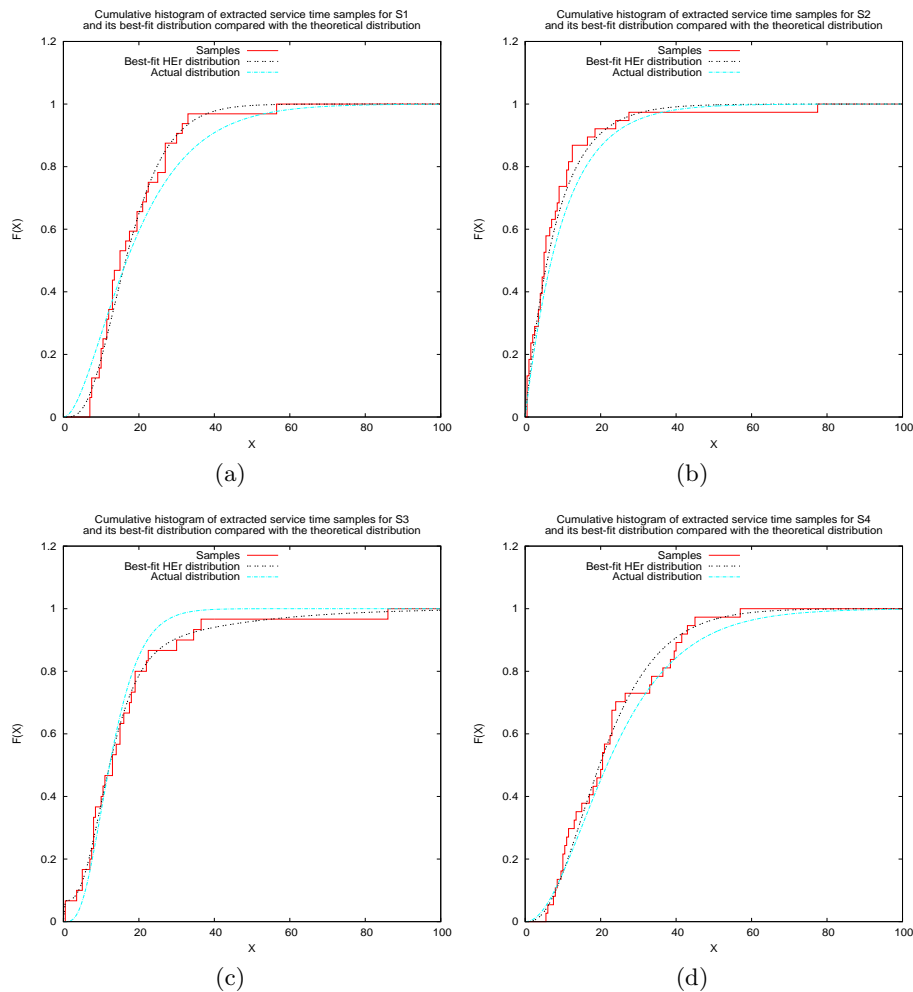


Fig. 7. Case Study 2: Graphs 7(a), 7(b), 7(c) and 7(d) show the cumulative histogram of the extracted service time samples (adjusted for synchronisation for S2 and S3) and its best-fit hyper-Erlang distribution compared with the theoretical distribution for S1, S2, S3 and S4 respectively.

4 Conclusion

This paper has presented a mechanism for synchronisation detection between customer-processing service areas in a system. This mechanism has been implemented as a new component of our existing methodology which automatically constructs GSPN performance models from location tracking data. We conjecture that our methodology can be applied to a large variety of systems whose underlying GSPN structure includes extended free choice (EFC) nets. An exam-

S1	Test Statistic	0.1268	
	α	0.1	0.05
	Critical Values	0.2076	0.2307
	Compatible ?	Yes	Yes
S2	Test Statistic	0.1074	
	α	0.1	0.05
	Critical Values	0.1914	0.2127
	Compatible ?	Yes	Yes
S3	Test Statistic	0.0861	
	α	0.1	0.05
	Critical Values	0.02141	0.2378
	Compatible ?	Yes	Yes
S4	Test Statistic	0.0921	
	α	0.1	0.05
	Critical Values	0.1938	0.2153
	Compatible ?	Yes	Yes

Table 4. Kolmogorov-Smirnov test at significance levels 0.1 and 0.05 applied to the extracted service time samples (adjusted for synchronisation) for each service area in the case study. The null hypothesis is that the extracted samples belong to the corresponding best-fit HErD.

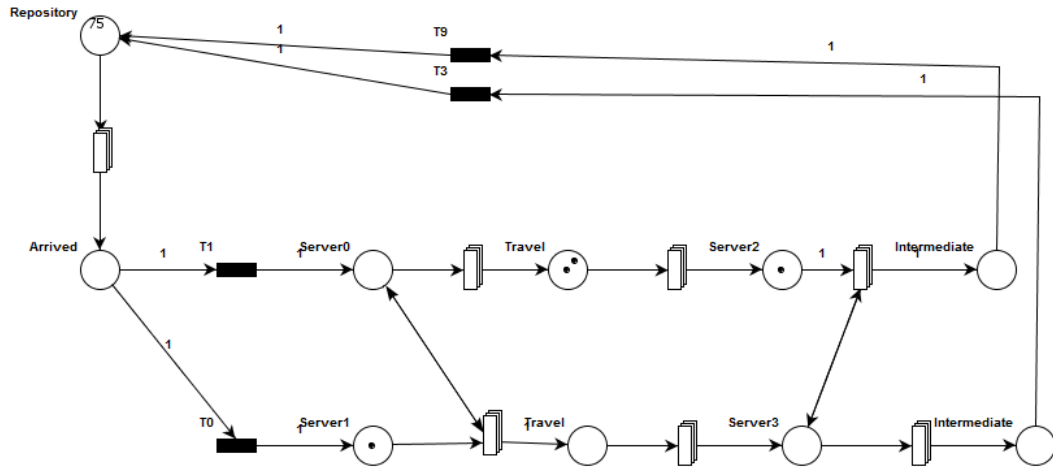


Fig. 8. Visualisation of the inferred GSPN performance model for the case case study (in compact transition form).

ple of a real-life situation where this methodology could be successfully applied is a Magnetic Resonance Imaging (MRI) unit of a hospital. The MRI control room is physically separate from the MRI chamber and requires a radiologist to operate it; the patient screening process cannot initiate if a radiologist is not present in the control room.

The case study results indicate that the developed methodology can infer the stochastic features and the presence of synchronisation in simple systems accurately, at least when synthetically-generated location tracking data is used.

Our current work has made several assumptions, i.e. one customer class, single-server semantics and random service discipline. In our future work we wish to relax them. We aim to use Coloured Generalised Stochastic Petri Nets (CGSPNs) to enable the support of multiple customer classes as well as prioritised service disciplines. Using CGSPNs we could also improve the accuracy of our models since we can use transitions that change the colour of the token (upon their firing) and thus control the routing of customers as they pass through various processing stages.

References

1. N. Anastasiou, T.-C. Horng, and W. Knottenbelt. Deriving Generalised Stochastic Petri Net performance models from High-Precision Location Tracking Data. In *Proc. 5th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2011)*, Paris, France, May 2011.
2. F. Bause and P. Kritzinger. *Stochastic Petri Nets*. Friedrich Vieweg & Sohn Verlag, 2002.
3. J. Billington, S. Christensen, K. V. Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In *International Conference of Applications and Theory of Petri Nets Proceedings*, pages 1023–1024, 2003.
4. P. Bonet, C. Llado, R. Puijaner, and W. Knottenbelt. PIPE v2.5: A Petri Net Tool for Performance Modelling. In *23rd Latin American Conference on Informatics (CLEI2007) Proceedings*, San Jose, Costa Rica, October 2007.
5. H. Bozdogan. Model selection and Akaike’s Information Criterion (AIC): The general theory and its analytical extensions. *Psychometrika*, 52:345–370, 1987.
6. Y. Fang. Hyper-Erlang Distribution Model and its Application in Wireless Mobile Networks. *Wireless Networks*, 7:211–219, 2001.
7. T.-C. Horng, N. Anastasiou, and W. Knottenbelt. LocTrackJINQS: An Extensible Location-aware Simulation Tool for Multiclass Queueing Networks. In *Proc. 5th International Workshop on Practical Applications of Stochastic Modelling (PASM 2011)*, Karlsruhe, Germany, March 2011.
8. T.-C. Horng, N. Dingle, A. Jackson, and W. Knottenbelt. Towards the Automated Inference of Queueing Network Models from High-Precision Location Tracking Data. In *Proc. 23rd European Conference on Modelling and Simulation (ECMS 2009)*, pages 664–674, May 2009.
9. M. Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri Nets for Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.
10. A. Thümmler, P. Buchholz, and M. Telek. A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE Transactions on Dependable and Secure Computing*, 3:245–258, 2005.