Supporting UML Sequence Diagrams with a Processor Net Approach

Tony Spiteri Staines
Department of Computer Information Systems, University of Malta, Malta
Email: toni_staines@yahoo.com

***Abstract*— UML sequence diagrams focus on the interaction between different classes. For distributed real time transaction processing it is possible to end up with complex sequence diagrams, containing messages related to system processes. It is difficult to examine alternative combinations of message passing. A solution is to translate these diagrams into an executable processor net model. This is based on the 'actor model', Petri net concepts and higher order net constructs. A case study taken from a flight reservation scenario is introduced and used to create a processor net model. This approach offers various advantages like identifying the main processes, executable model creation, verification, formalization, defining schemas and performance analysis.**

***Index Terms*— UML, processor net, Petri nets, modeling, verification, performance estimation**

I. Introduction

The UML notations have gained widespread use for designing different applications ranging from embedded systems [3],[10] to distributed information systems.

Requirements engineering is based on formal verification, modeling, performance estimation and performance engineering. These are based on different system view points and ideas like conceptual patterns in [25],[27]. It is difficult to cover all these issues with a single method. This could lead developers to opt for sub-optimal solutions. According to [36] the UML is more oriented towards software design whilst notations like Fundamental Modeling Concepts [33] focus on the conceptual system design, which is lacking in the UML. The Model-Driven Engineering approach also tries to tackle issues not accounted for in the UML in [26]. It is a fact that most software engineering methods lack the possibility of generating executable and verifiable models in [23].

The UML is divided into two main types of notations. These are static vs dynamic views. UML dynamic diagrams have various problems [6] and lack precise semantics. UML dynamic notations need verification. In [7] an algorithm verifying activity diagrams is proposed. Other suggestions deal with performance estimation and performance models based on the UML diagrams and semi markov processes.

A processor net can be used to model various aspects of the UML interaction diagrams.

A. Petri Nets, Higher Order Nets and the UML

Various classes of Petri nets exist. There are several proposals how the UML can be supported using Petri nets. Petri nets can be classified into two main types: i) basic Petri net classes and ii) higher order net classes.

Basic Petri nets offer a simple approach for validation, structural analysis and performance modeling. The token types are normally boolean and have a memory less state. Different proposals exist how to combine UML notations with these classes of Petri nets as in [1],[2],[4],[11]. It is possible to generate stochastic models like generalized stochastic Petri nets [22] from these Petri Nets.

Higher order nets [13],[14] have advanced properties. PrT nets, colored Petri nets [15],[18],[19],[20], algebraic Petri nets[34], object oriented Petri nets [1],[2],[17] and Petri nets composed of advanced data types etc. can be included in this category. Higher order nets are based on data tokens, parameter programming, arc inscriptions, input and output ports, procedures and functions. Some of these classes use functional languages as described in [8],[18]. It is possible to use these classes of Petri nets to model UML notations as is done in [1],[2],[15],[20]. Modeling of complex activities and communication can be easily achieved.

Unfortunately there is no general approach how to support UML notations with Petri nets. Communication based interaction diagrams seem to be the most suitable for conversion.

B. B, Z, VDM and the UML

UML notations are simple to use however they lack proper verification. Formal specification languages like B are used to tackle this problem. Unfortunately there are problems to translate UML into B language constructs addressed in [28]. There is no mechanism to 'lift off' a specification in B and some other specification languages. Z and VDM are schema based specification languages. Schemas define behavior to be promoted to

Based on " Supporting UML Sequence Diagrams Using a Processor Net Model", by T. Spiteri Staines which appeared in the Proceedings of the IEEE International Symposium and Workshop on ECBS 2007, Tucson Arizona, USA, March 2007. © 2007 IEEE.

higher levels. A schema can be defined in terms of other schemas. The processor net approach promotes the use of schemas for processors using VDM or Z as is done in [12]. Hence the problem when using B can be tackled.

C. Processor Net Model

The processor net based on the actor model in [12] is a higher order net that combines Petri net theory with other constructs. Instead of transitions, processors containing detailed instructions are used. This model allows the creation of different combinations. Petri nets allow the combination of components, without being restricted to a single general composition [34]. Places are defined as channels, transitions are defined as processors. Tokens are defined from different types, sets or colored sets. Tokens can be anything from boolean, integer, string to records or objects. Once basic types are created they can be used to build complex types. The processor net is constructed from an initial empty net obtained directly from a sequence diagram. This model is refined and processors are coded.

II. FLIGHT RESERVATION SCENARIO

A flight reservation system (FRS) is composed of different components. A local flight reservation system may connect to airline company databases or to global distribution systems. Global distribution systems can be accessed via different portals. The most important operations of FRS systems are: i) searching and viewing flight schedules, ii) Creating flight reservations, iii) modifying/ canceling reservations. A flight reservation is known as a PNR (passenger name record).

Flight reservation involves complex rules and different costs. There are complex i) functional and ii) non-functional requirements. E.g. the maximum time to create a transaction should never exceed three minutes. Certain airline companies send 'availability updates' to a central server for flight reservation every three minutes. There are operational issues e.g.: i) Once a Seat-Sell is submitted there is immediate booking. ii) Delete or modification update requires immediate notification.

The flight booking process is the most important operation. The reservation process involves many detailed steps and verification Different cost charges are associated with different processes and steps. E.g. a booking fee charge is more expensive than a query charge. GDS providers have different charging policies and service charges.

Some of the main GDS providers are: Amadeus, Sabre, Galileo and Worldspan. Most parts of these systems operate on legacy platforms. Emerging trends exist where companies like ITA software are offering optimized and flexible solutions using service oriented architectures and reconfigurable components.

A. Traditional Approach

A traditional approach is used in legacy based systems. The steps for reservation are carried out sequentially*: i) search request, ii) check seat availability, iii) check fare and iv) reserve flight.* These are depicted in Fig. 1.

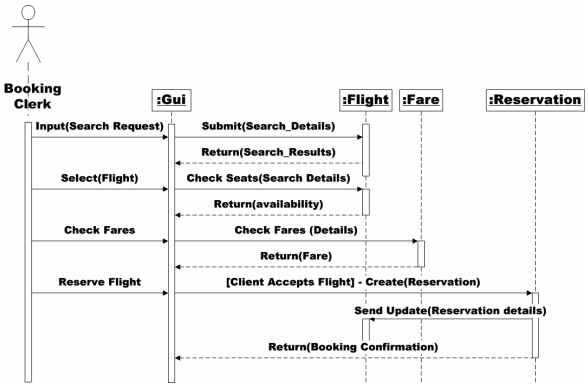

Figure 1. Normal flight reservation

B. Service Oriented Approach

In this scenario the steps for reservation can be carried out in different sequences or combinations. E.g. *search request* and *check seat availability* can be combined in a single step as in Fig. 2. It is possible for different classes to carry out different processes. Fig. 3. is a further modification of Fig. 2. where class :Flight instead of :Gui requests a *fare check* on receiving a search request.

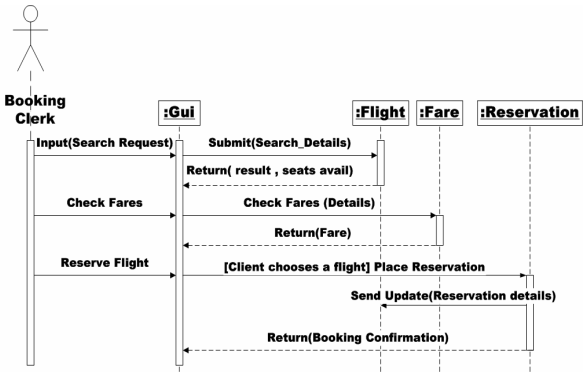

Figure 2. Flight reservation search + seat check combined

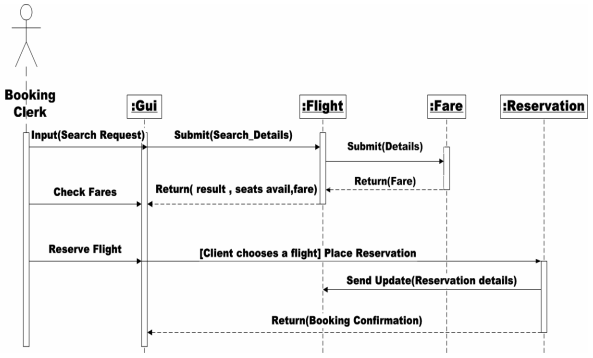

Figure 3. Flight reservation class :Flight requests a *fare check*

III. GENERATING AN EMPTY PROCESSOR NET

A. Sequence Diagrams

Sequence diagrams typically depict message passing between different classes and entities. UML 1.x and UML 2.0 sequence diagrams have certain differences.

Sequence diagrams have evolved from message sequence charts. Sequence diagrams can be converted to communication diagrams. In UML 2.0 sequence diagram message types are: synchronous or call, asynchronous, creation or reply. Sequence diagrams can employ complex constructs and become difficult to read.

B. Sequence Diagram Conversion

Fig. 1,2,3 describe different variations of the reservation process. This has to be performed within a defined period of time and some form of acknowledgement received.

The conventional approach to modeling sequence diagrams with Petri nets is to model message communication as in [11] or treat classes as processors.

A different 'main actor' based concept is presented here. The 'actor' initiates the whole process of message communication with the other classes via another main class like the **:GUI**. In this example the **:GUI** class is the identified actor class for the algorithm in fig. 4. This is because it is through the **:GUI** that the main processes are conducted. Processes of other classes are ignored because they do not concern the main actor directly.

Preconditions : A properly labeled UML 2 Sequence Diagram
Input: Sequence Diagram, identified class
Output: Processor – Net Diagram (Elementary Flat Processor Net Definition)

START

FOR EACH message m∈ M sent by the identified class to the right hand side exclusively
DO
Insert new processor p
Insert new channel c
Insert input flow/arc from c→ p
OD

FOR EACH return message rm∈ RM received by the identified class from the right hand side classes exclusively if there exists a match with a proper processor p
DO
Insert new channel c
Identify proper processor $p_n \in P$ where {n = 1.. total no of processors}
Insert output flow/arc from $p_n \rightarrow c$
OD

FOR EACH processor $p_x \in P$ where {x = 1 to | P |-1}
DO
Insert new channel c
Insert output flow/arc $p_x \rightarrow c$
Insert input flow/arc $c \rightarrow p_{x+1}$
OD

STOP

Figure 4. Sequence diagram conversion algorithm

The algorithm in fig. 4 is used to convert the UML 2 sequence diagram in fig. 1 into the empty processor net of fig. 5. The algorithm's function is to identify all important messages sent and received by the identified main actor class and generate a processor net. Other messages and constructs are ignored. The result is that the processor net model generated is a useful simplification of the sequence diagram. The same algorithm could be modified to generate a place transition Petri net.

The result of the algorithm in fig. 3 is defined as a flat net three tuple (P,F,C) model that is not decomposable. The three tuple (P,F,C) consists of a finite set of processors P where P = {p1,p2….p_n} and $P \neq \phi$, a finite set of directed flows/arcs F , where the arcs are from a processor p to a channel c or from a channel c to a processor p; $F \subseteq \{(CxP) \cup (PxC)\}$; $F \neq \phi$. A finite set of channels C, where C ={c1,c2,…c_n} , $C \neq \phi$. F,P,C are mutually disjoint. The empty process or net model shares properties with elementary nets.

The model obtained is still empty requiring additional development to obtain a fully executable processor net.

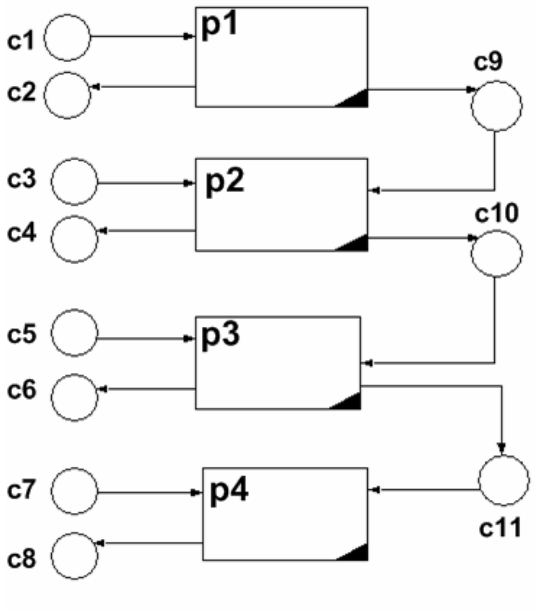

Figure 5. Initial processor net model

More rules and constructs are needed. Rules for tokens, processor firing, data types and complex class types are explained in [12].

IV. STEPS FOR CONSTRUCTING THE EXECUTABLE PROCESSOR NET

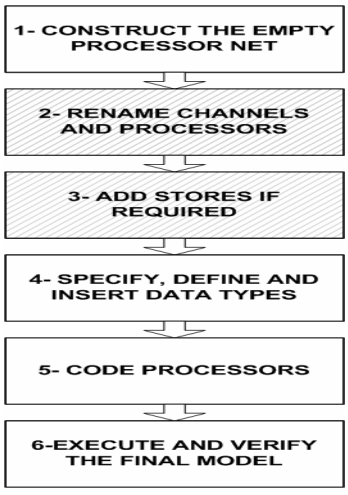

Figure 6. Steps for constructing the executable model

The steps in fig. 6 are used to build the fully executable processor net model. It is possible to use high level. High level / colored Petri net modeling tools like EXSPECT for executable specification described in [8],[9] or the CPN tools [15], [18] can be used for this task.

The model describe here was built and coded using the EXSPECT tool [8],[9] which is the ideal environment for the processor net. The main steps are described below.

Step1: Construct the empty processor net. The empty processor net in fig. 4 is built using channels, processors and connectors.

Step2: Rename channels and processors. This step is optional. The channels and processors are correctly named to represent the real system, making it more readable.

Step3: Add stores if required. Stores are similar to channels but they can have data types representing different system states and data similar to databases. Three stores are easily identified from the sequence diagram, these are: *Flight_Store, Fare_Store* and the *Reservation_Store* and have data types *Flight, Fare* and Reservation respectively.

Step4: Specify, define and insert data types. Data types must be correctly identified and defined. Different type constructors, sets, lists, mapping types and tuple types can be constructed. For the described system, types identified from the UML diagrams were used. A PNR (Passenger Name Record) type was created having details like *hename, flight_code, seats, hecontact, hetk, herf, heeot*. Other data types like flight ,fare, confirmation were also created. The tokens created are bound to these types.

Step5: Code the processors. Processor specification uses a functional specification language [8]. Special functions and constructs based on set theory can be specified. The following tasks are performed to code the processor: i) define inputs, ii) define outputs and iii) define a value expression for the processor. The input and outputs are the channels & stores. The processor definition or inscription processes the input tokens. The processor consists of a specification that builds the output value from the input value bound to the tokens. e.g. *out <— in* code copies the value of in channel to the out channel. Code in section 4 has been included as a comprehensive example.

Step6: Execute and verify the final model. The final model shown in fig. 7 is executed as follows: i) tokens with appropriate values are placed in the channels & stores, ii) processors are fired, iii) the output produced in the output channels & stores is checked. A processor is activated only when all its preconditions are met. This implies that its input channels & stores must contain information that satisfying the processor definition for activation. When a processor is activated the input tokens are consumed and output tokens are created and placed accordingly in the output channels.

Verification of the final model means that it is checked for dead processors, unreachable states, etc. Even if a processor fires and outputs something, this does not automatically imply that the processor is working successfully. A wrong result could actually be obtained. So verification means checking the results for correctness. If there are errors in the specification then it would mean that steps 4,5 would have to be repeated. Various scenarios composed of different properties can be examined. E.g. what happens when one important input value is missing. This is not possible with the UML sequence diagrams. The models could be enhanced with other constructs.

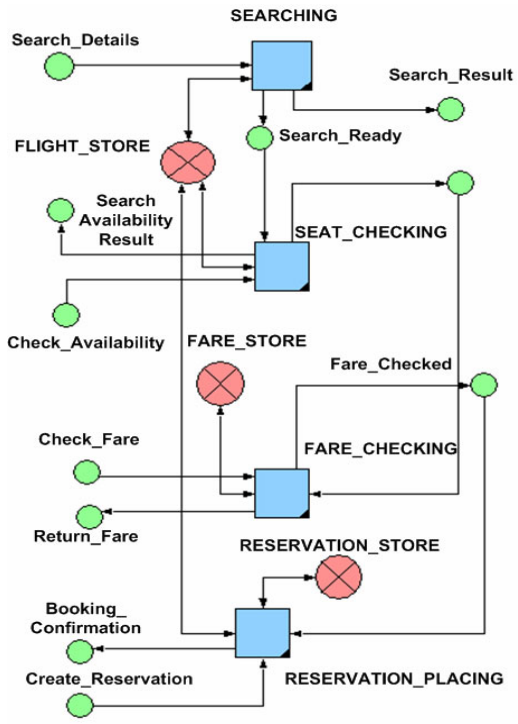

Figure 7. Final processor net

V. Formal Specification Schemas from the Processor Net

A. Specification languages

A formal specification language is mainly based on textual notations, whilst constructs for modeling like Petri nets, a processor net, colored Petri nets etc. are graphical. Two main parts of the processor net can be identified. i) the modeling part and ii) the specification as explained in [12].

Formal specification languages like VDM or Z are based on schemas. These languages are extendable. Complex types and functions can be defined in terms of basic types and primitive functions.

The processor net model can be properly described using elements from Z and VDM and extensions. Thus the processor net can be properly defined in terms of schemas. Schemas normally composed of [name| declaration| predicate] parts can be created for the processors, processor relations, channels, stores and the

complete system etc. It is also possible to define system constraints, states, initialization operations, exception handling, token firing and timing issues etc. in terms of schemas as in [21]. Processors being visible in processor net are ideal to use for schemas.

B. Processor reservation placing code

The code fragment below is a simplified version of processor: RESERVATION_PLACING.

```
result= (pick(set([x:Flight_Store|x@flight_code=
Create_Reservation@flight_code]))
// checks the reservation request with flight details

seat_check= result@seats>= Create_Reservation@seats
// checks for flight match with the reservation entered
and that seats are available .

update_flight_store= Flight_Store< [flight_code:result
@flight_code),avail:result@avail),seats:result@seats) -
Create_Reservation@seats]ins(set([x:Flight_Store|x@
flight_code!=Create_Reservation@flight_code]))
// update the flight_store contents with the reservation
details

--Processor main implementation--

if  seat_check= true then

    Reservation_Store<─Create_Reservation ins
    Reservation_Store,
    // add the new reservation to reservation store

   Booking_Confirmation<─ Create_Reservation,
    // send booking confirmation result

   update_flight_store

fi
```

C. Processor schemas

The processor implementation can be converted into high level schema.

Primitive functions defined are: i) a Projection function $\prod$ that selects values from rows and tuples (e.g. Π_2 (a,b,c,d) yields b , ii) a Set function that returns a set of members of x for a specific condition Set[x:T->bool]:$x, iii) a Pick function that converts a set into a tuple Pick[x:$T]:T [8,9,12]. An initial schema for the reservation placing is shown in fig. 8. This schema requires the support of other schemas that are not shown here.

Other schemas could be created for create_reservation, fare_checked, booking_confirmation. The processor relations for reservation placing and the other processors can be defined in a schema like format refer to [12] for more details. Schemas can be defined in terms of other schemas. This could be done for the complete system.

Logics, temporal logics and CCS can be used to describe the system.

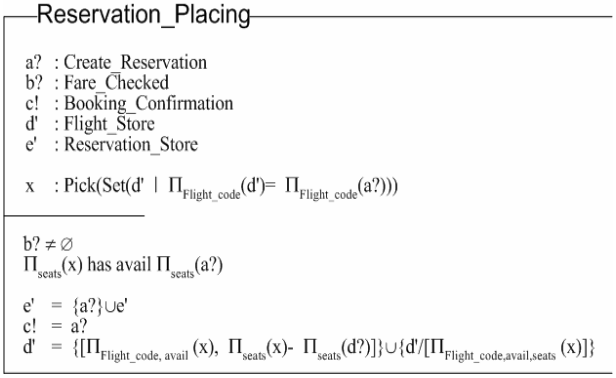

Figure 8. Schema for reservation placing processor

VI. MODEL CHECKING, PERFORMANCE ESTIMATION AND OPTIMIZATION METHODS

A. Model checking

The processor net is composed of higher order constructs applied to Petri nets. This means that it also has a simple identity. Simulation of the model will not necessarily indicate any structural problems. Prior to simulation it is possible to treat the model in fig.7 as a simple Petri net. Token identities are ignored, processors are treated as transitions. Channels and stores are treated as places.

Classic Petri net theory is well founded and has many analysis techniques applicable to structural checking. These techniques are based on mathematical properties. The model in fig. 7 can be represented using an incidence matrix or flow matrix. This would describe the input and output flows of the net. From the incidence matrix itself conclusions about the structure of the model can be deduced.

The reachability tree, reachability graph and place invariants can be found for the model. Other properties like boundedness, safeness, liveness, reversibility, etc can be examined. Some of these results are presented in the results section.

B. Processor net transformation method

The processor net in fig. 7 is based on acyclic behavior. The processor net can be transformed for time analysis.

The processor net is considered to be composed of a set of ordered processes executing in a particular order. Model modification is as follows: i) all channels data types are replaced by a single common data type ii) channels that not connecting the processors are removed iii) processors are modified to contain time, iv) a random store is added giving tokens random time values from a range $[e_i^-, e_i^+]$. The values e_i^- and e_i^+ are the min, max processing time. v) Optionally a firing cycle can be added.

Concepts from Petri net theory [29] are used. These ensure there are place and transition invariants and

feedback mechanisms [31,32]. The processor net can be directly transformed into a time place transition net.

C. Rules for alternative combinations

Processor net processors are ordered sequentially. For finding alternative combinations the following rules apply: i) Only two types of processor combination are possible: sequential or parallel. ii) The processor net is always choice free and conflict free. iii) All processors are functional. iv) The processor net has at least one directed path from the initial processor to the terminating processor. v) All processors must be in the directed path. vi) Precedence constraints for the processors are observed. Combining processors cannot result an increase in the number of steps in the system.

A large sized net can be reduced or simplified by combining processors and channels using Petri net reduction rules that preserve liveness and boundedness [12,32]. These rules restrict the number of possible combinations. Combining processors also depends on the business rules. These two steps indicate how to find combinations i) Define the transformed processor net. ii) Find alternative configurations using the given rules. If two processors a,b execute in parallel and $time(P_a) < time(P_b)$ then the critical path or cycle time is determined by processor b. So for time analysis processor 'a' could be omitted [12].

D. Performance estimation

Techniques focusing on single measure like time can be used for performance evaluation. The transformed processor net can be represented as an acyclical directed bipartite graph or activity network. Time analysis and critical path analysis can be performed. It is possible to add a cycle to the net and use cycle time analysis as is done for Time Petri nets. Simulation models for different configurations can be built and results compared.

E. Optimization Methods

An optimization problem can consist of a set of independent variables, parameters and conditions or restrictions defining acceptable values for the variables.

It is possible to treat alternative processor net model configurations as an optimization problem. In this case variables, conditions and restrictions not considered in performance estimation can be considered. The processor net is composed of a number of processors. Each processor can have various properties having direct or indirect relationships with other properties. The complete system would be composed of properties and conditions.

An optimization problem or *problem function* to be minimized can be described in terms of an *objective function F*, subject to *constraint functions* $\{c_i\}$. Both functions are real-valued scalar functions.

The category of optimization problems can be expressed in terms of a generalized NCP (non linear complimentary problem) form as follows: minimize $F(x)$, subject to $c_i(x)=0$, $i=1,2,\ldots,m'$; $c_i(x) \geq 0$, $i= m'+1,\ldots,m$, where $x \in \mathbb{R}^n$ [37]. This notation could be used even if it is not an NCP problem.

The steps to solve the optimization problem would be i) find which class fits the problem based on the *problem function* and *constraint functions* ii) find an optimum solution. The following properties can be used to identify the problem class: i) *objective function:* single variable functions, linear functions, sum of squares of linear functions, quadratic functions, sum of squares of non linear functions, smooth nonlinear functions, etc. ii) *constraint functions:* no constraints, simple bounds, linear functions, sparse linear functions, non linear functions, etc. E.g. a problem can have a linear objective function subject to non linear constraint functions. This opens the possibility of having many different problem classes.

Where the *constraint functions* and the *objective function* are both linear, linear programming can be used for optimization. Solving the problem might generate unique solutions.

A basic approach to optimizing the processor net is to try to minimize the system cost. This is based on the individual processor cost per unit time and the time utilized by that processor as in (1).

$$\sum_{p=1}^{n} C_p T_p \ . \qquad (1)$$

In (1) p = 1..n, where p is the processor number and n the maximum number of processors , C_p is the unique unit time cost of processor p ,T_p is the average processing time for processor p. If C_p the unique unit time cost is a constant for each processor, the objective and constraint function are linear then the problem can be represented as follows: minimize $c^T x$ subject to $Ax \geq b$, where $x \in \mathbb{R}^n$, c is a constant arbitrary non zero n- vector and x is the optimal solution as explained in [37]. There are different ways of formulating this problem for solving e.g. as an integer program, linear integer program etc.

The optimization models can range from simple to advanced. This depends on the variables and constraints involved. Complex optimization could be required. The problem can be formulated as a combinatorial optimization problem (COP). Solutions could be found using integer programming or heuristic algorithms [30].

From different perspective network calculus described in [38] can be used to model the network behavior for different scenarios. Network calculus fits the processor net idea under certain new assumptions. E.g. the processors are treated as network nodes etc. Estimation results are possible for bounds and calculating the QOS (quality of service). The processor net in fig. 7 can be classified as a feed-forward network. It has unidirectional links and no cycle. Since there is no loop, this system would have finite burstiness. The critical load factor could be used for calculations. A service curve can be constructed. This approach could be useful for response optimizing the overall system response time etc.

VII. RESULTS

A. Structural properties and reachability Analysis

To examine these properties the processor net model in fig. 7 is treated as a Petri net, token identities are ignored. The structural properties of the net can be inferred by examining the incidence matrix for the flows of the net and the reachability graph. The marking M_0 represents the token values in the channels and stores where M_0 = (c1..c11,s1,s2,s3), where s1, s2, s3 represent the marking of the stores and c1..c11 represent channel markings. The stores are never empty from the Petri net point of view. They can contain a token that is empty. The firing sequence is denoted by M_0-p_1-M_1-p_2-M_2-p_3-M_3-p_4-M_4.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 9. Flow matrix/ incidence matrix

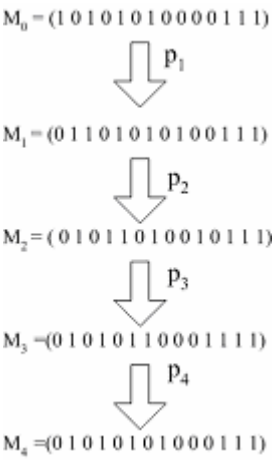

Figure 10. Reachability tree

The reachability tree in fig. 10 indicates that every processor is live, the net is bounded and safe. The net has home states. The result is interesting because it may help in the analysis of behavioral properties of complex systems. There are identifiable patterns in the flow matrix in fig. 9 indicating sequential behavior, boundedness. These indicate the structure of the net. It is possible to compute the complete net behavior using the flow matrix and the initial marking.

B. Modeling the behavioural sequence using real data

This modeling is done at a high level and the tokens contain real data. Processors can contain detailed programming logic that is useful for testing an executable model.

Appropriate data is placed in the stores, data tokens that represent actions are added and processors are enabled and fired getting results.

Processor RESERVATION_ PLACING is enabled by placing data tokens in Flight_Store, Create_Reservation and Fare_Checked. These are shown in the 'before' column in Table I. After firing the processor updated details are placed in the Flight_Store. The number of seats available are reduced by the number that was

TABLE I.
DATA FOR PROCESSOR RESERVATION PLACING EXECUTION

STORES	BEFORE	AFTER
Flight_Store	{[avail:'YES',flight_code:'KLM105',seats:150]}	{[avail:'YES',flight_code:'KLM105',seats:148]}
Reservation_Store	{}	{[hename:'NM1 SMITH J',flight_code:'KLM105',seats:2,hecontact:'AP92949693',hetk:'TKTL01MAR' herf:'RF JOE',heot:'ER']}
CHANNELS		
Create_Reservation	hename:'NM1 SMITH J',flight_code:'KLM105',seats:2,hecontact:'AP92949693',hetk:'TKTL01MAR' herf:'RF JOE',heot:'ER'	
Fare_Checked	1	
Booking_Confirmation		hename:'NM1 SMITH J',flight_code:'KLM105',seats:2,hecontact:'AP92949693',hetk:'TKTL01MAR' herf:'RF JOE',heot:'ER'

booked. A new reservation message is added to the Reservation_Store. Finally a reply message is placed in the Booking_Confirmation channel.

All the processors are executed and similar results are obtained.

C. Performance analysis

The idea of performance analysis is explained in section VI. Fig. 11 shows two different configurations. In configuration A two processors SEAT_CHECKING and SEARCHING have been placed to operate in parallel. Searching & seat_checking can be reduced to a single processor yielding identical results for this type of analysis only. This is not possible with every GDS.

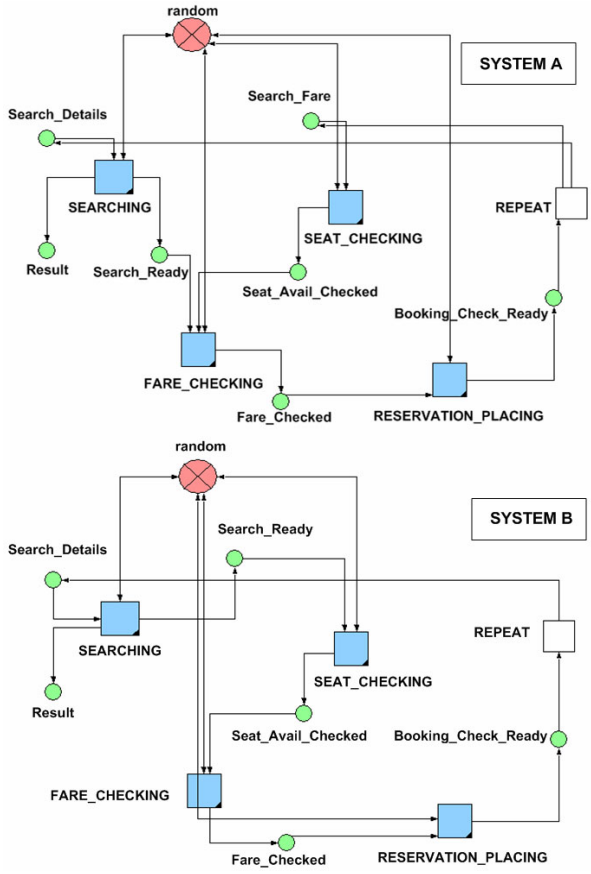

Figure 11. Two different configurations

TABLE II.
PROCESSOR MAX AND MIN TIME

PROCESSOR	Min Time (s)	Max Time(s)
SEARCHING	2	9
SEAT_CHECKING	2.9	10
FARE_CHECKING	2	8
RESERVATION_PLACING	20	40
REPEAT	0	0

TABLE III.
CUMULATIVE RESERVATION TIME

No. of System Cycles	Time Config. A	Time Config. B
10	373	418
30	1179	1308
70	2848	3166
100	4099	4565
150	6289	6994
200	8307	9251
250	10447	11610
300	12597	13977

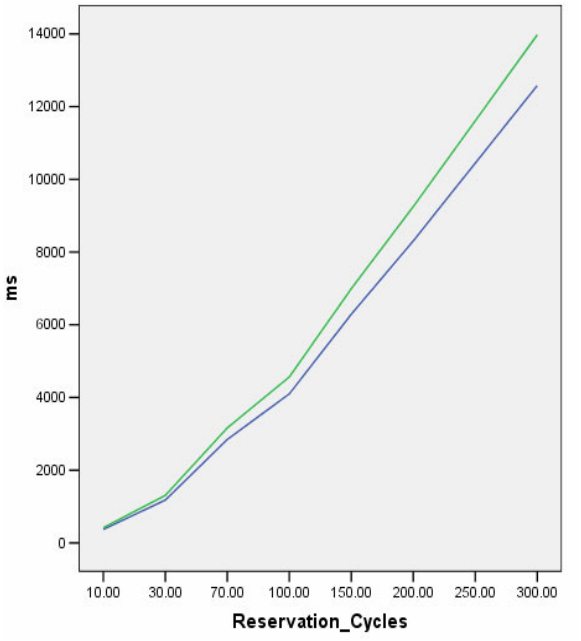

Figure 12. Cumulative reservation time

Modern service oriented architectures and component based architectures allow for different configurations. An example of this is QTX developed by ITA software.

Configuration A is an improvement over configuration B. The average cycle time is 42 seconds for configuration A and 46 seconds for configuration B. This is obtained from table III. The data was obtained from 300 system cycles, using the time ranges in table II. Fig. 12 depicts the results of table III graphically. Time values are generated randomly within the defined range from a uniform distribution. Table II shows the max and min times for each processor. The models in fig. 11 can also be analyzed using Petri net theory.

D. Optimization Example

Assuming that a different cost can be identified each processor per unit time. E.g. the estimated cost for searching per second is $ 0. 001333, fare checking is $ 0.0005, etc. It is possible to represent this as an *objective* function for minimization. The *objective function* for four processor model in fig. 7 is expressed in (2).

$$P = \frac{1.33a + 0.5b + 1.666c + 6.666d}{1000}. \quad (2)$$

Variables a,b,c,d represent the time in seconds consumed by each processor. Example *constraint functions* are presented in (3). These constraints are based on timing issues.

$$\begin{aligned} a &\geq 2 \\ b - 2a &\geq 0 \\ c - 1.5b &\geq 2 \\ b + c - 3a &\geq 0 \\ 1.5a + 2d &\geq 50 \end{aligned} \tag{3}$$

Combining two processors into one processor gives a different scenario. This could be possible for the UML sequence diagram in fig. 3. In this case the *objective function* for minimization can be represented as in (4), subject to the constraints in (5).

$$P = \frac{1.33a + 2.1b + 6.666c}{1000}. \tag{4}$$

$$\begin{aligned} a &\geq 2 \\ b - 2a &\geq 0 \\ 1.5b &\geq 2 \\ b - 3a &\geq 0 \\ 1.5a + 2c &\geq 50 \end{aligned} \tag{5}$$

The Simplex method [37] developed for solving linear programming problems can be used. The following results are obtained. The optimal solution for (2) is p = 0.174504 when a = 2, b = 4, c = 8, d = 23.5. This implies that the whole process would cost $0.174 and take 37.5 seconds. The optimal solution for (4) is p = 0.171776 when a = 2, b = 6, c = 23.5. This is minimally less expensive at $0.171 and takes less time. Modifying the constraints in (3) and (5) it is possible to obtain totally different results.

VIII. CONCLUSION

It has been shown how UML sequence diagrams can be converted to a processor net and analyzed. This is a practical way to support UML Sequence Diagrams. It is possible to use the processor net to obtain optimized models. This would be useful for describing special classes of real time systems.

The processor net model is more compact than a Petri net. It can be converted into other classes of Petri nets. Specifications can be defined for each processor. Formal definitions and proofs could be used. The processor net can be developed to create a detailed 'Actor Model' [12].

This approach also raises some fundamental issues if it is best to model the complete system before performing optimization. Optimization might imply a major system change.

The case study described in reality is more complex there are many issues like price to seating relations which include the time dimension. These are hard to describe and represent.

It is possible to find other analogies to the processor net approach. The processor net preserves properties that are common with other network structures of different types. It could be possible to apply concepts from these areas to the processor net or vice-versa.

Finally there are some limitations of this approach. i) The described algorithm cannot be used for all UML sequence diagrams. ii) the resulting net is a simplification of the UML diagram, this might be undesirable. iii) complex nets and even complex scenarios might be difficult to optimize, especially if proper information is not available.

REFERENCES

[1] L. Baresi, M. Pezze, "On Formalizing UML with High-Level-Petri Nets", *Concurrent object-oriented programming and petri nets: advances in Petri nets*, Springer-Verlag, ISBN 3-540-4192-x, 2001, pp. 276-304.

[2] L. Baresi, "Some Hints on Formalizing UML with Object Petri Nets", *6th World Conferenece Proceedings on Software Engineering and Knowledge Engineering*, Ischia Italy, 2002.

[3] G. Booch, *Object-Oriented Design with Applications*, 2d ed. Reading, Mass.: Addison-Wesley, 1991.

[4] B.Bordbar, L. Giacomini, D.J. Holding, "Design of Distributed Manufacturing Systems using UML and Petri Nets",*6th International Conference on Intelligent Systems*, Federation of Automatic Control(IFAC) Spain, May 2006, pp. 91-96.

[5] C.Canevet, S.Gilmore, J. Hilliston, L Kloul, P. Stevens, "Analysing UML 2.0 Activity Diagrams in the Software Performance Engineering Process", *DEGAS Project*, Lab. for Foundations of Computer Science Univ. of Edinburgh Scotland, Sep 2002.

[6] G. Engels, J. H. Hausmann, R. Heckel, S. Sauer, "Testing The Consistency of Dynamic UML Diagrams", *Proceedings of the 6th International Congress of Integrated Design and Process Technology* , IDPT, Pasadena CA USA, Jun 2002.

[7] R. Eshuis , R. Wieringa, An Execution Algorithm for UML Activity Graphs, *Lecture Notes in Computer Science*, Vol 2185, Springer-Verlag, 2001, pp. 47-61.

[8] Exspect User Manual, Technische Universiteit, Eindhoven, 1999.

[9] Exspect Tool, Technische Universiteit, Eindhoven.

[10] G. Frick, B. Scherrer, K.D. Muller-Glaser, "Designing the Software Architecture of an Embedded System with UML 2.0",*Proceedings of UML 2004 Workshop On Software Architecture Description & UML*, Portugal, Oct 11-15 2004.

[11] T. Gehrke, U.Goltz, H. Wehrheim, "The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process", *Technical Report Informatik-Bericht 11/98*, University of Hildesheim, Germany, Sep 1998.

[12] K.M. van Hee, *Information Systems Engineering A Formal Approach*, University Press Cambridge, 1994.

[13] K. Hoffmann, T. Mossakowski, "Algebraic Higher- Order Nets: Graphs and Petri Nets as Tokens",*16th International Workshop WADT*, Lecture Notes in Computer Science, Vol. 2755, Springer – Verlag, , Nov 2003, pp. 253-267.

[14] J.W. Janneck, R. Esser, "Higher-Order Petri Net Modelling – Techniques and Applications", ACM International Conference Proceedings Series; Vol. 145 Proccedings of the conference on application & theory of Petri Nets: Formal Methods in Software Engineering and defense

systems – Vol 12. (WSEFM), Adelaide Australia, Jun 2002, pp. 17-25.
[15] J.B. Jørgensen, "Coloured Petri Nets in UML-Based Software Development – Designing Middleware for Pervasive Healthcare" *,Proc. 4th Int. Workshop on the Practical use of CPN & CPN tools,* Diami Technical Report PB-560, Aarhus Denmark, Aug 2002, pp. 61-80.
[16] C. Kabajunga, R. Pooley, "Simulating UML Sequence Diagrams", *Proc. of 15th UK Perf. Eng. Workshops*, Dept. of Computer Science Univ. of Bristol, Jul 1999,pp. 23-33.
[17] B. Krena, T. Vojnar, "Type Analysis in Object-Oriented Petri Nets", *Proceedings of ISM'01 Hradec nad moranici Czech Republic* (Marq Ostrava), 2001, pp. 173-180.
[18] L.M. Kristensen, S. Christensen, K. Jensen, "The Practioner's Guide to Coloured Petri Nets", *International Journal On Software Tools for Tech. Transfer (STTT)*, Vol. 2, Springer-Verlag,1998, pp. 98-132.
[19] L.M. Kristensen, K. Jensen, "Specification and Validation of an Edge Router Discovery Protocol for Mobile AD Hoc Networks", *Lecture Notes in Computer Science*, Vol. 3147,Springer – Verlag,2004, pp. 248 -269.
[20] L.M. Kristensen, J.B. Jorgensen, K. Jensen, "Application of Coloured Petri Nets in System Development", *Lecture Notes in Computer Science*, Vol. 3098 , Springer-Verlag, 2004, pp. 626-685.
[21] D. Lightfoot, *Formal Specification using Z*, Palgrave , NY, 2001, ISBN 0-333-76327-0, Ch 6 pp. 37 – 97.
[22] C. Lindemann, A. Thummler, A. Klemm, M. Lohmann, O.P. Waldhorst, "Performance Analysis of Time-Enhanced UML Diagrams Based on Stochastic Processes", *3rd ACM Workshop on Software & Performance Rome* , Jul 2002, pp. 25-34.
[23] J.W.S. Liu, *Real-Time Systems*, Pretence Hall, NJ, 2000.
[24] R. Pooley, "Using UML to Derive Stochastic Process Algebra Models", *Proc. of 14thUK Perf. Eng. Workshops,* Univ. of Edinburgh Scotland, Jul 1998.
[25] J.A. Saldhana, S.M. Shatz, Z. Hu, "Formalization of Object Behavior and Interaction From UML Models",International Journal of Software & Knowledge Engineering, 11(6), 2001, pp. 643-673.
[26] D.C. Schmidt, "Model-Driven Engineering*", IEEE Computer Model-Driven Engineering\\Smart Cameras*, Vol 39 No 2, Feb 2006, pp. 25-31.
[27] C.U. Smith, Performance Engineering of Software Systems, Addison-Wesley, 1990.
[28] C.Snook, M.Butler, "UML-B: Formal Modeling and Design Aided by UML", *ACM Transactions on Software Engineering and Methodology*, Vol 15 No 1, Jan 2006, pp. 92–122.
[29] G. Stemersch, R.K. Boel, "Structuring acyclical Petri Nets for Reachability Analysis and Control", *International Journal of Intelligent Control and Systems*, Vol 10 No 2, Jun 2005, pp. 175-187.
[30] L.A. Wolsey, *Integer Programming*, Wiley, NJ USA, 1998.
[31] K. Yamalidou, J Moody, M. Lemmon, P. Antsaklis, "Feedback Control of Petri Nets Based on Place Invariants", *Technical Report of the ISIS Group University of Notre Dame IN 46556*, ISIS-94-002, 1994.
[32] M. Zhou, K. Venkatesh, *Modeling, Simulation, and Control of Flexible Manufacturing Systems A Petri Net Approach*, Series in Intelligent Control and Intelligent Automation vol. 6 World Scientific, MA USA, 1999.
[33] A. Knopfel, B.Grone, P. Tabeling, *Fundamental Modelling Concepts*, Wiley, NJ USA, 2006.
[34] J. Desel, E. Kindler, "Petri Nets and Components extending the DAWN approach", D. Moldt (ed.): Workshop on Modelling of Objects, Components, and Agents., Aarhus Denmark, Aug 2001.
[35] S. Donatelli , M. Ribaudo , J. Hillston, "A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets", Proceedings of the 6th International Workshop on Petri Nets and Performance Models, October 03-06, 1995,pp.158-173.
[36] P. Tabeling, B. Grone, "Integration Architecture Elicitation for Large Computer based Systems", ECBS 2005, Greenbelt, MD, USA, Apr 2005,pp.51-61.
[37] P.E. Gill, W. Murray, M.H. Wright, *Practical Optimization*, Academic Press, Harcout Science & Technology San Diego USA, 2000, pp.1-5, 67-77.
[38] J.Y. La Boudec, P. Thiran, *Network Calculus – A Theory of Deterministic Queing Systems for the Internet*, LNCS Vol 2050, Springer-Verlag, 2001 , ISBN 978-3-540-42184-9. Ch.1 & Ch.6.