Pervasive Service Creation using a Model Driven Petri Net based Approach

Achilleas Achilleos and Kun Yang
Dept. of Computing and Electronic Systems
University of Essex
Colchester, CO4 3SQ, United Kingdom
Email: {aachila, kunyang}@essex.ac.uk

Nektarios Georgalas and Manooch Azmoodech
IT Futures Research Centre
British Telecom Group Chief Technology Office
Ipswich, IP5 2YE, United Kingdom
Email: {nektarios.georgalas, manooch.azmoodeh}@bt.com

Abstract— **Service creation is a complex process that involves service analysis design, implementation and testing. Traditionally, the service is validated at the late stage of testing. This increases development costs since any necessary amendments would require an iterative improving cycle on service design and implementation; until the desired result is eventually reached. This paper proposes a service creation methodology and tooling with a twofold contribution: (i) based on its design, a service is validated early on and prior to implementation, (ii) the service code is automatically generated out of the validated service designs. To achieve this, our approach integrates model-driven architecture (MDA) with Petri Nets (PN). MDA is used to define the (i) Information, (ii) Service Oriented Petri Net (SOPN) and (iii) User Interface modelling languages, which support the service design and implementation phases. Petri Nets facilitate the service validation phase through the use of the SOPN language. By merging the two techniques we obtain a systematic and cost-effective approach for the creation of pervasive services. Concluding the methodology is applied in practice for the creation of a Flight Itinerary booking service.**

Keywords-Pervasive service creation; Petri nets; Model-driven

I. INTRODUCTION

The necessity to create agile services rapidly drives the computing world to seek new effective software engineering methodologies. Due to the advancements and diversity in technologies currently present, development of pervasive services becomes an increasingly complicated task. Pervasive services are highly dynamic in nature due to the diversities in the types of devices, the users and their increasing requirements [1]. Pervasive services demonstrate increased mobility since they should be able to run on any device, at any time and with minimal user attention. Another important characteristic of pervasive services is adaptability. Due to changes in hardware and software resources available in an environment the service requires to be able to adapt to the new conditions.

Such a service requires a more sophisticated software engineering approach to facilitate its creation process. Service creation is one of the most important phases in the service life cycle since it establishes the effectiveness and agility with which services are being developed. Generally the service creation process can be decomposed into the following individual tasks: (i) requirements analysis, (ii) design, (iii) implementation and (iv) validation through testing of the service implementations [2]. The downside of this approach is that the service is not validated until late in the process. If amendments are necessary, respective changes should be carried out in design and implementation, in an iterative cycle, until eventually a valid service is developed. Sometimes even, in the interest of expediting delivery, changes are only made in the code and not in the designs, introducing thus a discrepancy between design, documentation and implementation. This is not very cost-effective and impacts on efficiency.

Our proposed approach to service creation introduces service validation straight after the service design phase, deferring implementation until later. In other words, the service is validated through its design and not through its implementation. Therefore, any changes detected are handled early on and directly in the design specification. After certain iterations and as soon as the design is stabilised meeting exactly the service requirements, the implementation phase is entered where the code is automatically generated out of the design specification. This approach saves sufficiently in development resource as (i) it reduces implementation time through automatic code generation and (ii) it defers implementation by excluding it from the validation iterative cycles. The approach furthermore ensures that design specifications and implementation are fully synchronised.

The methodology is facilitated by MDA and Petri Nets. MDA [3] descended from the Unified Modelling Language (UML) and aims at separating application reasoning from the underlying implementation technology. Keeping the approach at an abstract level avoids dealing with platform specific implementation concerns. It also allows to produce rigorous design specifications and to generate technology-specific implementations. In order to validate service behaviour and ensure service correctness, our approach uses Petri Nets. PN theory [4] comprises a formal technique, suitable for modelling service behaviour, due to its simple graphical representation and its concurrent and asynchronous nature [5]. Furthermore describing a service using a PN model allows the qualitative analysis and validation via execution of its dynamic behaviour.

By integrating MDA with PN theory our approach provides a rigorous design, validation and code generation process. In the context of this paper we tackle the pervasiveness of a service from the viewpoint of generating diverse implementations from an abstract service model. With MDA we keep the service design and implementation phases at an

978-1-4244-2202-9/08/$25.00

abstract level and with Petri Nets we aid the validation phase. In this way we are able to obtain an abstract and formal specification of the service, validate its integrity and automatically generate the executable service implementation.

The remainder of this paper is organised as follows. Initially Section 2 presents related research work on the field and following Section 3 describes the core of the paper, which is the service creation methodology. Section 4 describes the modelling languages of the methodology and Section 5 illustrates the methodology in practice through a realistic example application. Section 6 discusses the results of the approach and its application and identifies future work.

II. Related Work

This paper is primarily associated with the work carried out on service creation using high-level effective software engineering methodologies. Several researchers attempted to devise systematic methodologies for the rapid creation and delivery of advanced services. In an example of the conventional approach, Adamopoulos et al. [2] described service creation as the most abstract and important stage of the service life-cycle. Their approach proposes an iterative service creation process, which is based on successive development and refinement of a telematic service. Therefore when the implementation phase is completed, the validation is performed via testing to identify any problems. Subsequently the implementation is refined accordingly and these steps are repeated until eventually the desirable implementation is achieved. This approach, although well formed, it does not tackle the necessity to generate implementations for diverse platforms and leaves validation at the implementation phase where technological aspects are introduced. Additionally it relies on multiple sub-processes something that hinders the approach increasingly complex.

Andonoff et al. [6] used Petri Nets with Objects (PNO) formalism to model Workflow Web Services (W2S). The objective was to graphically and formally describe W2S and derive OWL-S specifications automatically. The approach provides the ability to gracefully design and verify the workflow service using the PNO specification. Furthermore it facilitates automatic generation of OWL-S specifications, which provide a semantic Web accessible format that enables W2S publication. However, OWL-S is quite complex in nature with a cumbersome grammar. Moreover the capability to transform PNOs to other specifications or implementations is not provided as it is required for pervasive services. Manual development of supporting tools is also a downside.

The necessity to formalise process behaviour for semantic web services was stated by Kohler and Ortmann [7]. They affirm the necessity to have a formalised and visual technique for the representation of web services, which provides means for refinement. They introduced Service Description nets to capture the semantics of web services and provided the mapping to Algebraic nets as defined in [8], to benefit from their analysis techniques. The use of Algebraic nets assures the executability of the service nets. Although this auspiciously offers the benefit of a formal and graphical technique; it deals only with the specification of web services.

Distinct research work by Roch H. Glitho et al. [9] focused on the creation of Internet telephony services. This approach uses a service creation environment (SCE) for defining telephony services using pre-defined graphical Java components. Subsequently from the service design a Java based implementation is generated via the use of the SCE. The authors justly support that a high-level SCE is necessary and benefits non experts, since they can work at a high-level of abstraction without requiring technology-specific knowledge. The drawback is that once again validation is kept after the implementation phase, resulting in costly iterations until the desirable implementation is obtained. Moreover the SCE generation capability is tailored around Java and the SCE is manually developed from scratch. In this paper we attempt to overcome these issues by proposing a platform-independent framework and an approach for pervasive service creation.

III. A Model Petri Net Based Methodology

Combination of the model-driven technique with the PN formalism provides the capability to define an effective and systematic service creation methodology. The model-driven technique provides the potential to develop a supporting framework for the methodology. Petri Nets on the other hand serve as the formal specification of service functionality, i.e. the service tasks, and facilitate service validation.

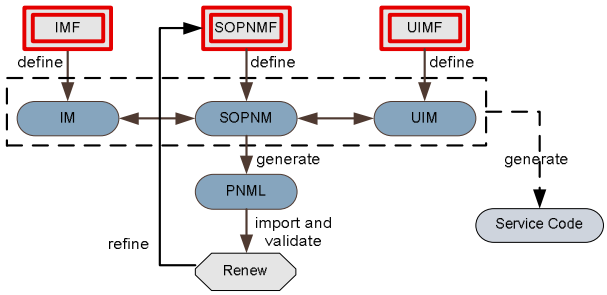

Figure 1. Model-driven Petri Net based methodology.

Figure 1 presents the approach that allows the analysis, design, validation and code generation of services. Primarily the core Service Oriented Petri Net (SOPN) modelling language was defined as an MDA metamodel. It facilitates the definition of service behaviour in the form of a PN which can then be validated. Furthermore two complementary modelling languages were defined; Information and User Interface. These languages are used by designers to define respectively the service information objects and graphical user interfaces that complement the service behaviour. Each language is accompanied by a dedicated graphical editor, used to produce the design specifications, and dedicated code generators. SOPN is specifically accompanied by two code generators. In the validation phase, the SOPN code generator transforms the SOPN-expressed behaviour to the Petri Net Markup Language (PNML); explained later in more detail. The PNML format is used then by the RENEW Petri-Net tool [10] to validate the service behaviour model. In implementation phase the languages' code generators produce the executable code of the service from the three platform independent models.

The Integrated Eclipse Modelling Environment (IEME) [11] model-driven capabilities facilitate the automatic generation of the languages, their editors and the generators as distinct modelling frameworks. These are namely the Information modelling framework (IMF), the SOPN modelling framework (SOPNMF) and the UI modelling framework (UIMF). The frameworks are generated in the form of Eclipse plug-ins and are integrated into the IEME to compose the model-driven Petri Net based framework used for supporting pervasive service creation.

The metamodel definitions of the aforementioned languages were produced using the concept of meta-modelling as presented in [12]. According to this, a metamodel represents a language's abstract syntax, concrete syntax and semantics. The Eclipse Modelling Framework (EMF) and the Graphical Modelling Framework (GMF) capabilities provided by the IEME facilitated the metamodel definition in practice.

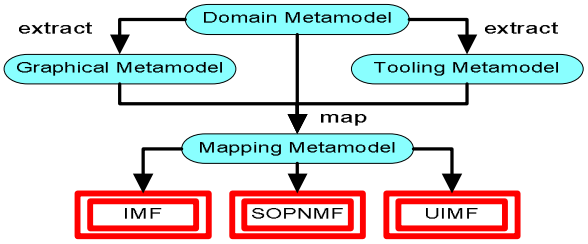

Figure 2. Developing the domain specific modelling frameworks.

Figure 2 illustrates the procedure of defining each language and its dedicated editor. The domain metamodel includes the semantics of the language and is defined using the abstract syntax provided by the EMF meta-meta language; an implementation of the Meta Object Facility (MOF). From the domain metamodel the graphical and tooling metamodels are automatically generated, which represent the concrete syntax of the language. These metamodels include the corresponding graphical elements and the palette tooling of the editor to be generated. Mapping the distinct metamodels we obtain a mapping metamodel that includes all the necessary artefacts. From the mapping metamodel we automatically generate a distinct modelling framework for each language.

Initially, service analysis and design, are performed using the three devised modelling languages. Via these languages we define Platform Independent Models (PIMs). These models capture service requirements and provide a precise service definition. Subsequently we proceed to the realization of the service validation phase. In this phase the SOPN model must be validated to guarantee the completeness and correctness of the service model. To achieve this step we have developed a template based extensible generator that is able to transform the SOPN model to an equivalent PNML representation. The transformation is carried out using the openArchitectureWare (oAW) feature of the IEME. Using the model and the generator we execute the transformation with the aid of the oAW workflow engine.

An excerpt of the SOPN-to-PNML generator template is presented in Figure 3 revealing the one-to-one mapping of SOPN model transitions to the PNML representation. Template based generators are defined using the oAW xPand language. The language provides the capability to refer to elements of the domain metamodel, acquire the values from the model and transform them into a text-based representation. Specifically the keyword *FOREACH* points to the entire set of transitions defined in the model. Graphical properties such as *position* and *dimension* are required to enable the redraw function, when the resulted PNML-expressed model is imported into the PN tool, which will eventually validate the service behaviour. Additional semantics generated, such as *type* and *value,* are also imperative to the execution of the SOPN model.

```
«FOREACH net_transitions AS transition»
<transition id= "«transition.id»">
«IF transition.transition_expression != null»
<«transition.transition_expression.type»>
<graphics> <offset x= "«transition.transition_expression.offset_x»"
          y="«transition.transition_expression.offset_y»"/> </graphics>
          <text> «transition.transition_expression.element_value» </text>
</«transition.transition_expression.type»>
«ENDIF»  <graphics> <position x="«transition.possition_x»"
          y="«transition.possition_y»"/>
          <dimension x="«transition.dimension_x»"
          y="«transition.dimension_y»"/>
          <fill color="«transition.fillType_color»"/>
          <line color="«transition.lineType_color»"/>
          </graphics> </transition>
«ENDFOREACH»
```

Figure 3. Extract of the PNML template generator.

The PNML syntax is a well recognised and widely accepted standard supported by a variety of Petri Net tools. Consequently when the PNML file is generated, the model is imported into the PN tool. At this stage the service model behaviour is validated via dynamic execution. In case the execution halts, it is signified that the service process model is erroneous. Hence the SOPN-represented model is further refined and undergoes the service validation phase again. Conversely if the service execution process is carried out successfully, the model integrity is assured. Thus the service validation phase is effectively complete and we subsequently proceed to the service implementation phase.

The implementation phase is accomplished by transformation of the PIM models to executable service code. Intermediate transformation of PIMs to Platform Specific Models (PSMs) can be executed to ease the transition from models to code as shown in [1]. Prior to generating the implementation from the models we impose further implementation specific constraints onto the models; using the Object Constraint Language (OCL). This step is carried out to guarantee that the executable service code to be generated is not an erroneous one. To perform the code generation phase we developed code generators in the form of templates as it is illustrated in Figure 3. These generators aim to deliver executable service code from the Information, UI and SOPN models; instead of PNML syntax. A specific programming language (e.g. Java, C#, EJB) is selected and the code generators are tailored and built around that implementation technology. Although the PIM models do not require being altered, different code generators are needed for transforming the models to different programming languages. This is the last stage of the methodology from which the executable service code is generated and being deployed onto the chosen platform.

IV. THE MODELLING LANGUAGES

A. Information model

The Information model represents different *Entities* of the service with their corresponding *Properties*. *Properties* can be either structural or behavioural denoting respectively *Attributes* or *Operations*. Figure 4 represents the metamodel definition, which is similar to an Entity-Relationship (ER) diagram and can be used to define abstract object models. Every entity has a set of attributes and operations that characterize the entity. Primarily an entity has a specific *name* declaration that denotes the name of the entity. Additionally each entity's attribute or operation has a corresponding *name* and *type* declaration, which denote the name and type definition of the attribute or name and return type definition of the operation. Examples of such attributes' type declarations are: *String, Boolean, Integer* and *Array*.

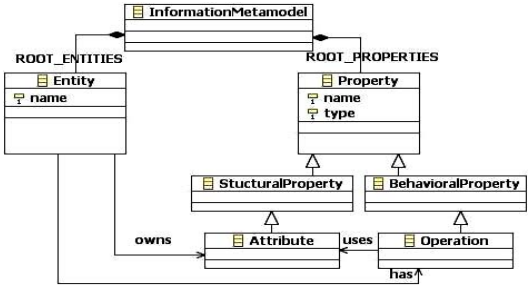

Figure 4. Information model metamodel definition.

B. Service Oriented Petri Net model

Petri nets are a widely accepted formalism particularly suited for modelling concurrency mainly due to the principle of locality [5]. Additionally PN provide both a graphical and algebraic representation something that makes them easily understandable. It is one of the most constructive languages available for modelling concurrent and distributed processes.

Definition 1 (Petri Nets): The base of all PN extensions is a Net structure $N = (P, T, F)$ where:

1) $P = \{p_1, p_2, ..., p_m\}$, where $m \geq 0$, is a set of *Places*.
2) $T = \{t_1, t_2, ..., t_n\}$, where $n \geq 0$, is a set of *Transitions*, $P \cap T = 0$.
3) $F \subseteq (P \times T) \cup (T \times P)$, is a flow relation for the set of *Arcs*.
   - $I : P \to T^{\infty}$, is the *Input Arc*, a mapping from places to bags of transitions.
   - $O : T \to P^{\infty}$, is the *Output Arc*, a mapping from transitions to bags of places.

Definition 2 (Markings/Tokens/Firing): A marking $m = \{p_1, p_2, ..., p_t\}$, is a mapping that assigns a number of tokens to each place. The holding of a condition is represented by a token in the corresponding place. Figure 5 shows the simplest possible net N, its initial marking $m_1 = \{1,0\}$ and the occurrence rule for the transitions using the example t_1 transition. A transition t is enabled and can fire in a marking m if $\forall p \in {}^{\bullet} t : m(p) \geq 1$, i.e. there is at least one token in each of its input places. Firing a transition removes tokens from its input places and deposits it to its corresponding output places, resulting to a new marking m'. In our example of Figure 5 the net marking becomes $m_2 = \{0,1\}$.

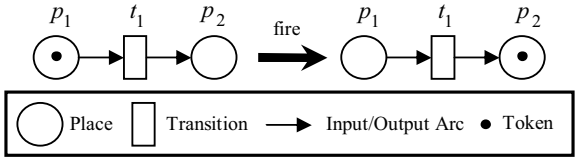

Figure 5. The basic Petri Net model.

The simplest extension to basic PN is the Place/Transition (P/T) nets, which introduce the arc weight function W. Consequently the PN structure becomes $N = (P, T, F, W)$. The input arc weight determines the number of tokens required at an input place to enable the firing of a transition. In contrast the output arc weight determines the number of tokens generated and deposited to an output place when an enabled transition fires. Several extensions [13] have been conceptualized to provide solutions to systems engineering.

One such extension is Reference Nets (RN) [10], an object-oriented Java based extension, which is well suited for modelling concurrent systems. The idea put forward in this paper is to make use of a required subset of RN to formulate the SOPN modelling language, which can facilitate the entire process of service creation. Keeping only the abstract properties of RN preserves the technology-neutral nature of the approach, avoiding technology specific implementation concerns. SOPN incorporates high-level concepts that provide the capability to define the service behaviour unambiguously. Definition 3 describes the semantics included in a SOPN model definition. These provide the formal definition of SOPN models and guide the service execution process. Furthermore each semantic property is mapped accordingly to a graphical representation.

Definition 3 (SOPN): The abstract RN structure $N = (P, T, F, D, E, W)$, where:

1) $D = \{d_1, d_2, ..., d_i\}$, where $i \geq 0$, is a finite set of primitive data types or objects that can be assigned as tokens to places.
2) $E = \{e_1, e_2, ..., e_j\}$, where $j \geq 0$, is a finite set of service execution control expressions.
3) $W = \{w_1, w_2, ..., w_k\}$, where $k \geq 0$, is a finite set of primitive data types or objects that can be assigned as weights to input and output arcs.

Figure 6 outlines the artefacts of the SOPN modelling language defined in the form of a metamodel. It extends the

implemented standard PNML Core metamodel [14], which defines the basic properties of a Petri Net. The Core metamodel is in fact at an abstract level and provides only the basis for implementing concrete Petri Net types; e.g. P/T nets. The SOPN metamodel is produced using the *import* and *inheritance* capabilities provided by EMF, to extend the abstract PNML Core metamodel concepts. This process is used as an alternative of the UML *merge* concept, since EMF does not natively support the *merge* concept.

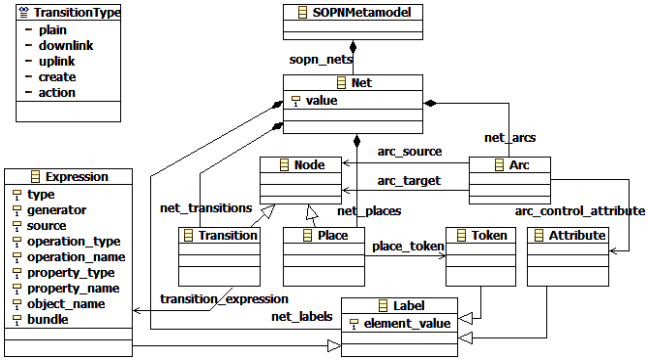

Figure 6. Service Oriented Petri Net model metamodel definition.

A SOPN model represents a single *Net* that has a specific *value* declaration and includes *Places* and *Transitions* interconnected using *Arcs*. Each *Place* is assigned a *Token* that is represented in the form of a *Label* with a corresponding *element_value*. Additionally each *Transition* is assigned an *Expression* which includes several declarations; e.g. *operation_type* that represents the return type of the method call and *operation_name* that represents the name of the corresponding method. The set of declarations that an expression possesses depends on the *TransitionType* enumeration, which reveals the type of the *Transition*. Furthermore an *Arc* is assigned an *Attribute* that is represented also in the form of a *Label* with a corresponding *element_value*. An arc has a specific *source and target Node*. A *Node* represents the parent class of *Places* and *Transitions*.

C. User Interface model

A service besides its objects and behaviour it additionally requires graphical user interfaces (GUIs) with which the user interacts in order to utilize the service. Therefore a corresponding abstract metamodel definition was devised for defining graphical user interfaces as illustrated onto Figure 7. Based on our practical experience in implementing graphical user interfaces we included in the metamodel only basic set of entities characterising the structure and operation of a GUI. Since the language is represented as a metamodel definition it is easily extensible and allows the developer to extend it including more entities, such as certain types of GUI components like combo-box, radio-button etc.

Referring to the metamodel concepts each *Component* element has a *name* and a *type* declaration and can contain other components. It additionally owns different properties *e.g.* a *Frame* can have a location *Property*. A *Property* element includes the *name* and *value* declarations. Furthermore a component can generate an event or implement an event listener. Both the *Event* and *Listener* elements have a unique *name* declaration. An event includes also a *source* property, which pinpoints to the component that generates that event.

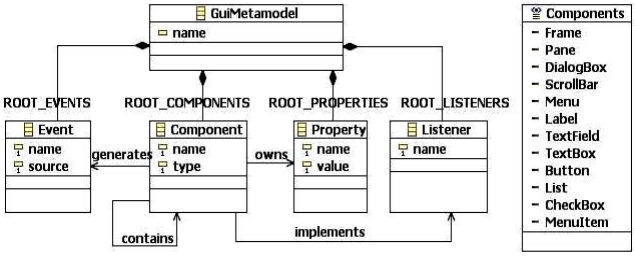

Figure 7. User Interface model metamodel definition.

V. A Realistic Pervasive Service

In this section we present a case-study that uses our approach in practice for the creation, i.e. specification, validation and implementation of a pervasive service. The pervasiveness of the service is depicted by generating diverse implementations from the same platform independent models.

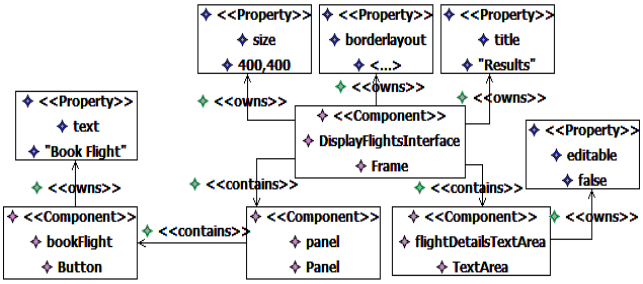

Figure 8. Flight itinerary booking example interface model.

Let's assume that Peter, who is the user of the service, forgot to make a reservation for a flight while he was at the office due to a very busy day. A reminder on his laptop device notifies him of the scheduled conference he will attend in two days. Despite being in a taxi on his way home, he can directly access the flight reservation service using his laptop to book his trip. Peter connects to the internet using his phone's GPRS, makes use of the service and completes his flight booking itinerary. An example user interface model of the service Peter used on his laptop device is illustrated in Figure 8. The model illustrates the main Frame component that owns three different properties. Properties are signified using keywords, which are recognised by the code generator. OCL constraints can be applied to check the validity of the keywords prior to transformation. Additionally the main Frame contains two child components, which also possess their corresponding individual properties. From the interface model the equivalent code was automatically generated. The resulting graphical user interface is illustrated in Figure 10.

Figure 9 presents the second PIM model of the service; the SOPN process model. At the start of the net we have initialization tokens in places *p1, p3, p5, p7, p9, p11 and p13*, which signify the start of the service execution process. Transitions *t1-t7* are named *creation* transitions since *t1* refers

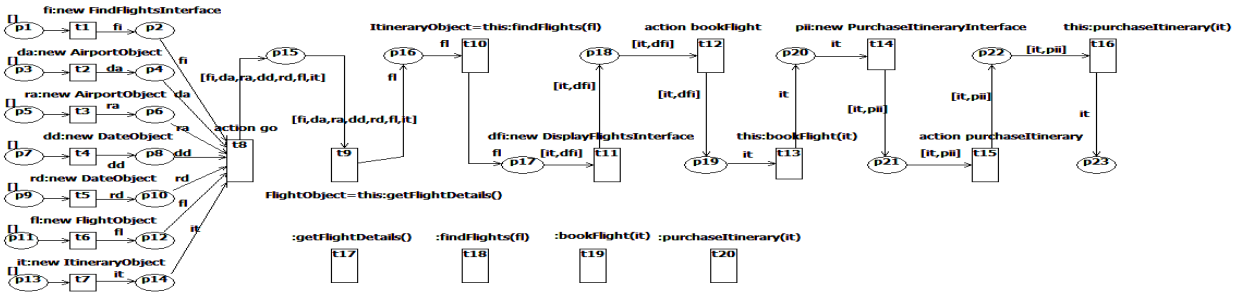

Figure 9. Flight itinerary booking SOPN process model.

to the creation of the initial interface *FindFlightsInterface* and *t2-t7* refer to the creation of the entities required to search for a specific flight. For instance transition *t2* refers to the departure airport object *da:new AirportObject.*

Figure 10. Flight itinerary booking service interface.

Transitions *t8, t12 and t15* refer to *action* transitions generated by the corresponding components. For instance the button *Book Flight* shown onto Figure 10 generates the action associated with transition *t12*. Additionally we have *downlink* transitions that represent operation calls and *uplink* transitions that enable the execution of the operation. Particularly uplink transitions point to the operation statements which we assume to be implemented manually. Furthermore each arc is assigned an attribute/weight that refers to a specific entity of the net and controls the service execution flow similarly to transitions.

VI. Conclusion And Future Work

In this paper we presented an effective model-driven Petri Net based approach for the creation of pervasive services. The effectiveness of the approach lies on the integration of the two techniques. Primarily, the model-driven paradigm keeps the approach at an abstract level avoiding implementation complexities. Secondly, Petri nets blend the essential service validation phase into the approach. Finally we established the applicability of the devised methodology by generating a realistic service. Forthcoming work aims in the improvement of the methodology by enhancing the modelling languages; i.e. through the imposition of metamodel level constraints.

Acknowledgement

The work presented in this paper is partly supported by British Telecom under the MOSE project and the UK Engineering and Physical Sciences Research Council (EPSRC) under project PANDA.

References

[1] K. Yang, S. Ou, M. Azmoodeh and N. Georgalas, “Policy-based model-driven engineering of pervasive services and the associated OSS”, BT Technology Journal, vol. 23, no.3, pp. 162-174, July 2005.

[2] D. X. Adamopoulos, G. Pavlou and C. A Papandreou, “Advanced Service Creation Using Distributed Object Technology”, IEEE Communications Magazine, vol. 40, no.3, pp. 146-154, March 2002.

[3] A. Kleppe, J. Warmer and W. Bast, “MDA Explained: The Model Driven Architecture: Practice and Promise”, Addison-Wesley, 2005.

[4] C. Girault and R. Valk, “Petri Nets for System Engineering: A Guide to Modelling, Verification and Applications”, Springer, 2003.

[5] R. Tan and S-U. Guan, “A Dynamic Petri Net Model for Iterative and Interactive Distributed Multimedia Presentation”, IEEE Transactions on Multimedia, Vol. 7, No.5, pp. 869- 879, October 2005.

[6] E. Andonoff, L. Bouzguenda and C. Hanachi, “Specifying workflow web services using Petri nets with objects and generating of their OWL-S specifications”, EC-Web, Copenhagen, Denmark, LNCS 3590, pp. 41-52, Springer, August 2005.

[7] M. Kohler and J. Ortmann, “Formal aspects for semantic service modelling based on high-level Petri nets*”, CIMCA-IAWTIC*, Vol. 1, No. 28-30, pp. 107-112, Vienna, Austria, November 2005.

[8] W. Reisig, “Petri nets and algebraic specifications”, In K. Jensen and G. Rozenberg, editors, High-level Petri Nets - Theory and Application, pp. 137-170, Springer, 1991.

[9] R. H. Glitho,F. Khendek and A. De Marco, “Creating value added services in Internet Telephony: An overview and a case study on a high-level service creation environment”, IEEE Transactions on System, Man and Cybernetics, Vol. 33, No. 4, pp. 446-457, November 2003.

[10] Reference Net User Guide, Theoretical Foundations Group, Dept. of Informatics, University of Hamburg, [Online] Available: http://www.informatik.uni-hamburg.de/TGI/renew/renew.pdf , 2006.

[11] A. Achilleos, N. Georgalas, K. Yang, “An Open Source Domain-Specific Tools Framework to Support Model Driven Development of OSS”, in ECMDA-FA, LNCS 4530, pp. 1–16, Springer, June 2007.

[12] J.P. Nytun, A. Prinz, M. S. Tveit, “Automatic Generation of Modelling Tools”, in ECMDA-FA, LNCS 4066, pp. 268–283, Springer, June 2006.

[13] C. A-Yahia, N. Zerhouni, A. E. Moudni, and M. Ferney, “Some Subclasses of Petri Nets and the Analysis of Their Structural Properties: A New Approach”, IEEE Transactions on Systems, Man and Cybernetics, Vol. 29, No.2, pp. 164-172, March 1999.

[14] E. Kindler, “Software and Systems Engineering – High-level Petri Nets: Part2 Transfer Format, Proposed Draft Addendum to International Standard ISO/IEC 15909 Part 2 – Version 0.6.3”, June 2005.