

Correctness Checking of Pervasive Behaviour by Mapping Task Models to Petri Nets

Estefanía Serral

Faculty of Economics and
Business, KU Leuven
Department of Decision
Sciences and Information
Management
KU Leuven, Naamsestraat 69
estefania.serralasensio
@kuleuven.be

Johannes De Smedt

Faculty of Economics and
Business, KU Leuven
Department of Decision
Sciences and Information
Management
KU Leuven, Naamsestraat 69
johannes.desmedt@kuleuven.be

Jan Vanthienen

Faculty of Economics and
Business, KU Leuven
Department of Decision
Sciences and Information
Management
KU Leuven, Naamsestraat 69
jan.vanthienen@kuleuven.be

ABSTRACT

Context-Adaptive Task models are a state-of-the-art executable modelling language to develop pervasive computing systems. Although these models have proven to be successful in the automation and support of user daily tasks, they do not provide a proper checking for ensuring the correctness of the designed systems. In this paper, we investigate and define mappings to translate the task models into Coloured Petri Nets (CPN), a formalism that provides powerful techniques for simulation and verification. By using these mappings, task models can be translated to their equivalent CP nets, enabling that the system's behaviour described in the task models can be exhaustively checked at design time to ensure a proper system execution at runtime.

Author Keywords

Pervasive Computing, Task Models, Petri Nets, mappings

ACM Classification Keywords

D.2.12. Interoperability: Data mapping; D.3.3 Language Constructs and Features

INTRODUCTION

Pervasive computing [9] is a computational paradigm in which technical systems are developed and deployed in the environment to support humans in their daily activities. To do this, these systems provide pervasive services that can effect changes in the environment and suggest appropriate activities to the users. A key requirement is supporting these activities by remaining safe for the system users.

Context-Adaptive Task models [12] are a state-of-the-art modelling language to represent daily routines that can be supported by pervasive environments. This language provides executable conceptual models that hierarchically specify the tasks a system should execute in a certain context in order to support a user in the conduction of her daily routines. Task models have been successfully adopted in model-driven pervasive infrastructures [12, 11, 13]; however, although these models have shown to be effective for supporting routine automation, they provide limited support for checking the correctness of the designed behaviour in order to assure a safe

interaction with the users. For instance, it is essential to ensure that the modelled assisting routines do not produce loops when they are executed or that the execution of several routines in sequence or in parallel does not produce undesired behaviour. This is important for pervasive environments such as smart homes, and is essential for critical environments such as Ambient Assisting living (AAL) facilities, hospitals, etc.

In order to provide support for this validation, this paper investigates how to map context-adaptive task models to Colored Petri Nets (CPN). CPN is a modelling language that has a graphical notation and a formal definition for the execution semantics; but what is more important is the support of CPN for powerful associated analysis techniques. CPN allows to validate: 1) structural properties, such as structural liveness (i.e., to validate that there is at least a context state where no deadlocks are found), repetitiveness (i.e., the routines should be able to be executed again provided the context that activates them is repeated), consistency (i.e., a certain context state always fires the same actions); and 2) behavioural properties, such as reachability of a certain state (e.g., to validate that the system can always return to a safe state), behavioural liveness (to validate the absence of deadlocks for certain context), coverability (there is no designed task that can never be executed).

Thus, by mapping task models to CPN, this work enables the visual simulation of the system execution and the automatic checking of the correctness of the designed behaviour.

The rest of the paper is organized as follows. The next sections explain: the related work; a running example; our baseline: task models and CPN; the mappings between both formalisms; and finally the discussion, future directions, and conclusions.

RELATED WORK

In this paper, we define mappings to transform task models into Petri nets, as such enabling the correctness checking of the pervasive behaviour described in the task models. Several works have defined mappings from different languages to Petri nets in order to take advantage of their simulation and verification techniques. For instance, in [7], the authors pro-

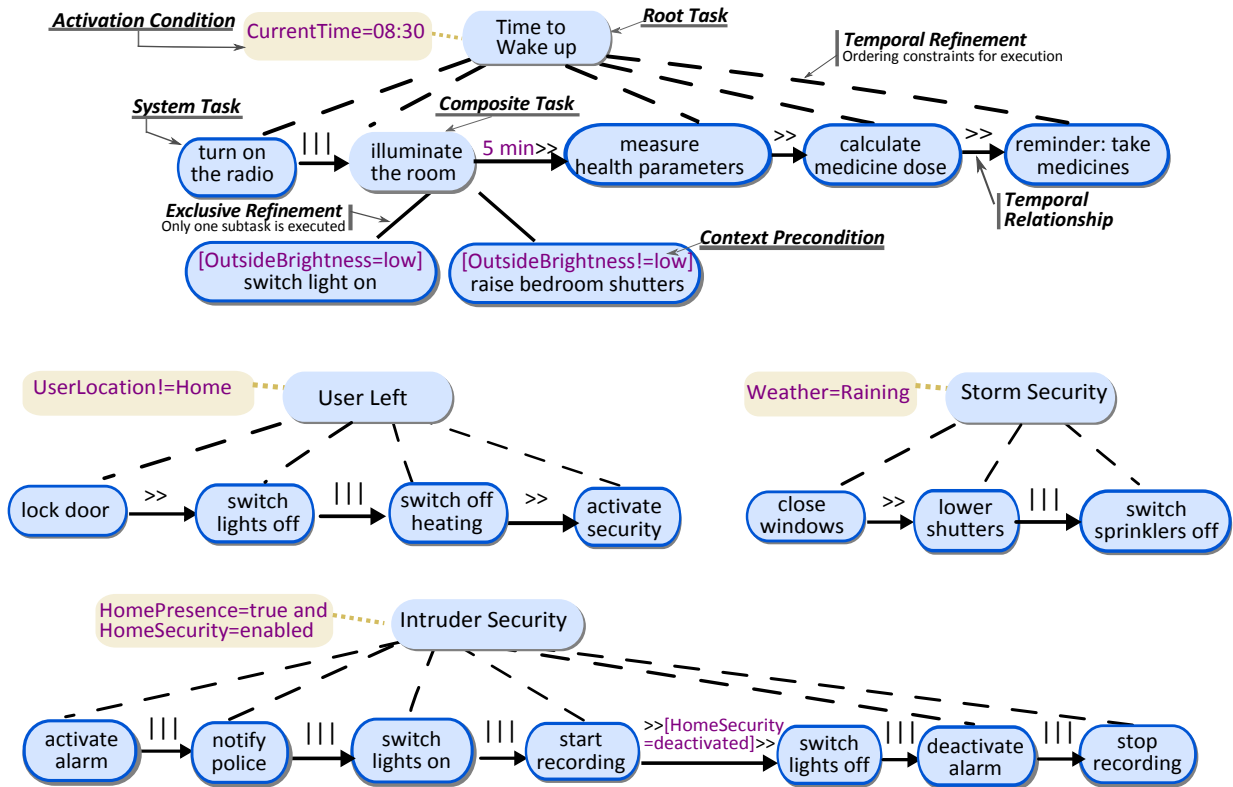


Figure 1. A task model representing the “Time to Wake up”, ”User Left”, ”Storm Security” and ”Intruder Security” user routines

pose a mapping from Business Process Execution Language (BPEL) onto WorkFlow nets, which are a subclass of Petri Nets (PN), to enable soundness checking. The paper presented in [15] also describes how UML 2 Activity diagrams can be intuitively translated into Petri net notations for execution and validation. However, no other previous work has focused on mapping task models onto Petri nets.

In addition, Petri nets have also been used for describing pervasive behaviour and checking the correctness of this behaviour. For instance, an outstanding example is the one presented in [6]. Using Petri nets, the authors model workflows to control the device operations and the interactions among the devices of a household. Thus, designers can run the PN simulation to predict environmental effects before (or while) a workflow is (being) executed. Also, in [1], the authors propose to use CPN to represent and validate the requirements of a pervasive system by means of animated simulation. In our work, the system is specified using task models, which are more comprehensive than Petri nets by users (e.g., doctors and nurses in the healthcare domain) and provide a more compact notation [5] [3]. Thus, in our work Petri nets are used to complement the task models for verification purposes, allowing the possibility to use the best model according to their benefits for the specific pursued goal.

RUNNING EXAMPLE

An AAL home pervasive environment supports the daily routines of an elderly person. The AAL home is equipped with Ambient Intelligence devices, controlled by pervasive ser-

vices [12] that monitor and collect context information, as well as enabling interaction with the users and the environment. Some of the supported routines are as follows:

- **Time to Wake up:** At 8:30 in the morning, the system turns on the radio and tunes to the user’s favourite channel and illuminates the room by raising the blinds if the brightness outside is high, or by switching the lights on otherwise. After 5 min., the system requires the user to measure their health parameters. With this information, the system calculates the medicine dose that the user needs to take and notifies his/her about the medicines and their dose that should be taken before breakfast.
- **User Left:** When the user leaves home, the system locks the door, switches off the lights and the heating, and activates the security.
- **Intruder Security:** When an intruder is detected (the security is enabled and someone’s presence is detected in the house), the system activates the alarm, notifies the police, switches on the lights and starts recording. Afterwards, when the situation is under control (the security is disabled), the system switches the lights off, deactivates the alarm and stops recording.
- **Storm Security:** when it starts raining, the system closes all the windows that are open, lowers the shutters so the windows are not stained, and switches off the sprinklers of the garden if they are working at that moment.

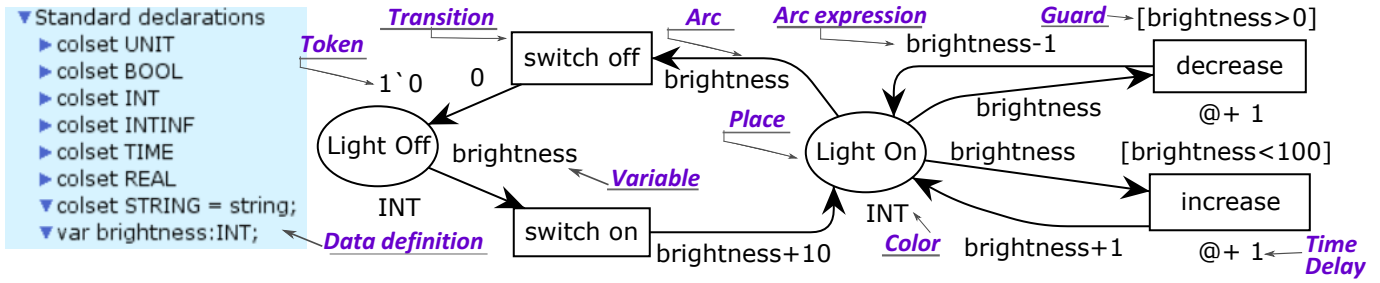


Figure 2. CPN to represent the behaviour of a light-dimming switch

RESEARCH BASELINE

This section presents our research baseline: Context-Adaptive Task Models (CATM) and Coloured Petri Nets (CPN).

Context-Adaptive Task Models (CATM)

Context-Adaptive Task Models (CATM) specify how a pervasive infrastructure can support its users in carrying out everyday activities [12]. CATM are inspired by Hierarchical Task Analysis (HTA) [14], which constructs a task tree that refines a high-level task into a set of executable ones.

A task model defines the routines that the system must automate to support user activities like the ones explained in the running example. Using HTA, each user routine is described as a task hierarchy. The *root task* represents the routine itself, and has an *activation condition* that indicates the situation in which the routine is activated. For instance, the "Time to Wake up" routine is to be executed at 8:30 a.m. The root task is iteratively broken down into simpler tasks by means of two task refinement constructs: *exclusive refinement* and *temporal refinement*. Exclusive refinement (represented by a solid line) decomposes a task into a set of subtasks in such a way that exactly one subtask will be executed. Temporal refinement (represented by a dashed line) also decomposes a task into subtasks; however, all the subtasks shall be performed following a specific order which is graphically depicted by the arrows between sibling tasks. Temporal constraints make use of Concurrent Task Trees (CTT) operators [8]. For example, in Fig. 1, we use:

- *Enablement* ($T_1 \gg T_2$): task T_2 is triggered when task T_1 finishes.
- *Enablement with Condition Constraint* ($T_1 \gg [s] \gg T_2$): after the completion of T_1 , T_2 is started as soon as the situation s holds.
- *Enablement with Time constraint* ($T_1 t \gg T_2$): after the completion of T_1 , T_2 is started as soon as the time period t has elapsed.
- *Parallel* ($T_1 ||| T_2$): T_1 and T_2 are executed in parallel.

The task refinement process ends when every leaf task in the tree can be associated with a pervasive service (controlled by the pervasive system), which is responsible for executing the task. For example, task "raise bedroom shutters" is executed through a pervasive service that controls the shutters' engine. Thus, the leaf tasks are called *System Tasks*, while the tasks

that are refined are called *Composite Tasks*. Also, a task can have a *context precondition* (depicted between square brackets before the name of a task) to indicate that the task is only executed if the precondition holds; otherwise the task is skipped. In an exclusive refinement, the task executed will be the one whose precondition is satisfied.

For a precise definition of the elements that can appear in a context-adaptive task model, its metamodel can be found in [13].

Coloured Petri Nets (CPN)

Coloured Petri Nets (CPN) [4] provide model checking capabilities such as state-space analysis, with emphasis on verifying behavioural and structural properties. CPN provide a high level of expressiveness; in this section, we focus on explaining the constructs needed for translating task models. As a simple example, Figure 2 shows a CP net that represents the behaviour of a light-dimming switch. As it can be seen in the figure, a CP net is a directed bipartite graph in which the nodes consist of: *transitions* (i.e. actions that may occur, represented by bars or a square), such as "increase" and "decrease"; and *places* (i.e. states, represented by circles), such as "Light Off", "Light On". Each directed *arc* (represented by arrows) connects either a place to a transition or a transition to a place.

Each place contains a set of markers called *tokens*, such as the marker "0" of the "Light Off" place. Each of these tokens carries a data value that can be modified by the occurring transitions. Each data value belongs to a given data type, which is called *color*; e.g., the token colours of the figure are INT, which stands for the integer type. An arc can have an expression, which is used to describe how the tokens change when the transitions occur. In the example CP net, the transition "switch on" moves the INT token's initial value 0, to the "Light on" place, increasing the brightness to 10.

A transition can occur if and only if its input places (i.e., each place having an arrow from it to the transition) have sufficient tokens, and these tokens match the arc expressions' colour set. For instance, at the beginning of the execution, only the transition "switch on" can occur, since only the place "Lights Off" contains a token. Also, transitions can have *guard* conditions (indicated between brackets next to the transition square) which must be satisfied to be able to fire the transition, such as the guard of "decrease" ($[brightness > 0]$), which prevents that brightness can be a negative number.

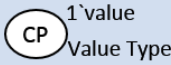
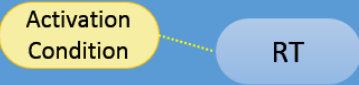


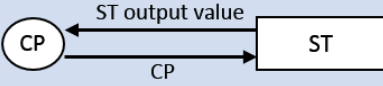


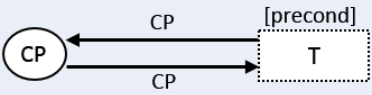
TASK MODEL CONSTRUCTS	COLORED PETRI NET CONSTRUCTS
Context Property <i>CP</i>	Variable and place with a token. The token has the value and the type of the context property. 
Routine Task <i>RT</i> (root task) 	Transition with a guard that corresponds to the context situation of the routine task. The transition corresponds to the first subtask of <i>RT</i> ; except in the case the first subtask has a parallel temp. rel, in which an auxiliary transition is then created. The transition must have an input and an output arc to each place that represents a context property used in the context situation. 
System Task <i>ST</i> 	Transition.  The service linked to the task is omitted since it will not be really executed for the model checking. However, if the task modifies, at least, a context property <i>CP</i> , this effect will be mapped to an input and an output arc from the transition to each one of the places that represent those context properties. The output arc will have the output value of the action as inscription.
Composite Task <i>CT</i> 	1. The composite task is omitted in the mapping and substituted by its subtasks. 2. Subpage Transition : all the subtasks of the composite task must be created in the subpage. 
Context Precondition [<i>precond</i>]	Guard of the transition <i>T</i> that represents the task where the precondition is defined. The transition must have an input and an output arc to each place that represents a context property used in the context. 

Table 1. Tasks and context data mappings from CATM to CPN

TASK MODEL CONSTRUCTS	COLORED PETRI NET CONSTRUCTS
<p>Temporal Refinement</p>	<p>If T is the Root Task: the temporal refinement is translated like if T was linked by an enablement temporal relationship to T1. If T is a Composite Task, then the refinement is not directly mapped to any element, but corresponds to the mapping of the composite task and its refined tasks.</p>
<p>Exclusive Refinement</p>	<p>Each refined task is mapped to a transition with a guard that contains the context precondition of the task. All the transitions will have an input arc that comes from the same place. Each transition must have an input and an output arc to each place that represents a context property used in the transition guard.</p>

Table 2. Refinements' mappings from CATM to CPN

When a transition occurs, exactly one token that matches the arc expression is moved from the input places to the output places (i.e., each place to which there is an arrow from the transition). If no transition can occur, then the net is said to be dead.

In CPN, *time delays* can also be specified to describe delays (indicated by @+t next to the transition square, where t is the delayed time), e.g., the "increase" and "decrease" transitions have a delay of 1 unit.

MAPPING CATM ONTO CPN

Tables 1-4 show the mappings each CATM construct to its corresponding constructs of CPN. Figure 3 shows the CP Nets that represent the three most complex routines of the running example created by applying these mappings. These routines cover the application of all the mappings.

As it can be observed in Table 1, each **Root task** is mapped to a place (shown in blue in Figure 3) connected to a transition with a guard; **system tasks** are mapped to transitions; **composite tasks** can be omitted or transformed to a subpage transition (the last option is recommended when there are more than two subtasks in the refinement in order to avoid the excessive growing of the resulting CP net); and **context preconditions** are mapped to guards. In addition, each **context property** is represented in CPN as a place with a token that has the initial value of the property. This idea is based on [6], which uses Petri nets to model pervasive behaviour. Thus, each transition that reads or modifies a context property must have an input and output arc from and to the place that represents that property. For instance, the

transition "lock door" in the "User Left" routine has as guard *UserLocation* <> "Home" and therefore is linked with the place "UserLocation" (see Figure 3).

Table 2 shows the mappings to transformed **refinements**. As it can be seen in the table, the temporal refinement is normally omitted, unless the upper task T is the Root task, in this case, the refinement is mapped like T was related by an *enablement* temporal relationship to the first subtask. The exclusive refinement is mapped to a CPN OR-split: a place related to several transitions where the token can go to only one of the transitions according to their guards. An example of exclusive refinement can be seen in the CP Net of the "Time to Wake up" routine (see specifically tasks "switch on bedroom lights" and "raise bedroom shutters" followed by the place "P17").

Table 3 shows the mappings to transform **temporal relationships**. They are generally mapped to places that connect the transitions (i.e., the tasks linked by the relationships) between each other. Each temporal relationship has its own mapping in order to appropriately reflect their different behaviour in CPN. For instance, the result of applying the Parallel relationship mapping can be seen in the "Intruder Security" routine shown in Figure 3: when an intruder is detected (see guard of T1), the token is placed in 4 different places (P4-P7) enabling the 4 activities that must be performed.

Finally, Table 4 shows the mappings for the comparative and Union operators, which are straightforward.

For simulation and model checking purposes, after applying

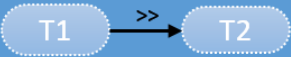
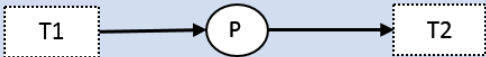
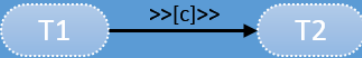
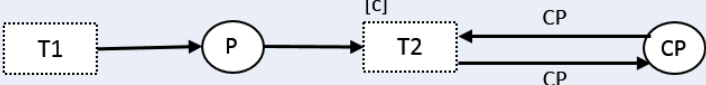
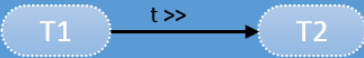
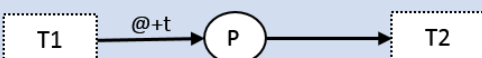

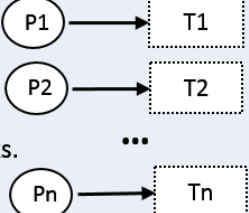
TASK MODEL CONSTRUCTS	COLORED PETRI NET CONSTRUCTS
<p>Temp. Rel. Enablement</p> 	<p>Place with an input arc from T1 and an output arc to T2</p> 
<p>Temp. Rel. Enablement with Condition Constraint</p> 	<p>Place with an input arc from T1 and an output arc to T2. T2 has a guard that contains the condition constraint c.</p>  <p>The transition T2 must have an input and an output arc to each place that represents a context property used in the guard.</p>
<p>Temp. Rel. Enablement with Time Constraint</p> 	<p>Place with an input arc from T1 and an output arc to T2. The input arc (or the output arc) of the place has a delay of t.</p> 
<p>Temp. Rel. Parallel</p> 	<p>For each task: A place with an output arc going to a transition that is connected by an arc to the next task that is not related by a parallel temporal relationship, or to the final place if there are no more tasks.</p>  <p>In the case there is a previous task, it should be connected to all the created places (P1, P2, ..., Pn).</p>

Table 3. Temporal Relationships' mappings from CATM to CPN

TASK MODEL CONSTRUCTS	COLORED PETRI NET CONSTRUCTS	TASK MODEL CONSTRUCTS	COLORED PETRI NET CONSTRUCTS	TASK MODEL CONSTRUCTS	COLORED PETRI NET CONSTRUCTS
=	=	<	<	AND	,
!=	<>	>	>	OR	orelse

Table 4. Comparative and Union Operators mappings from CATM to CPN

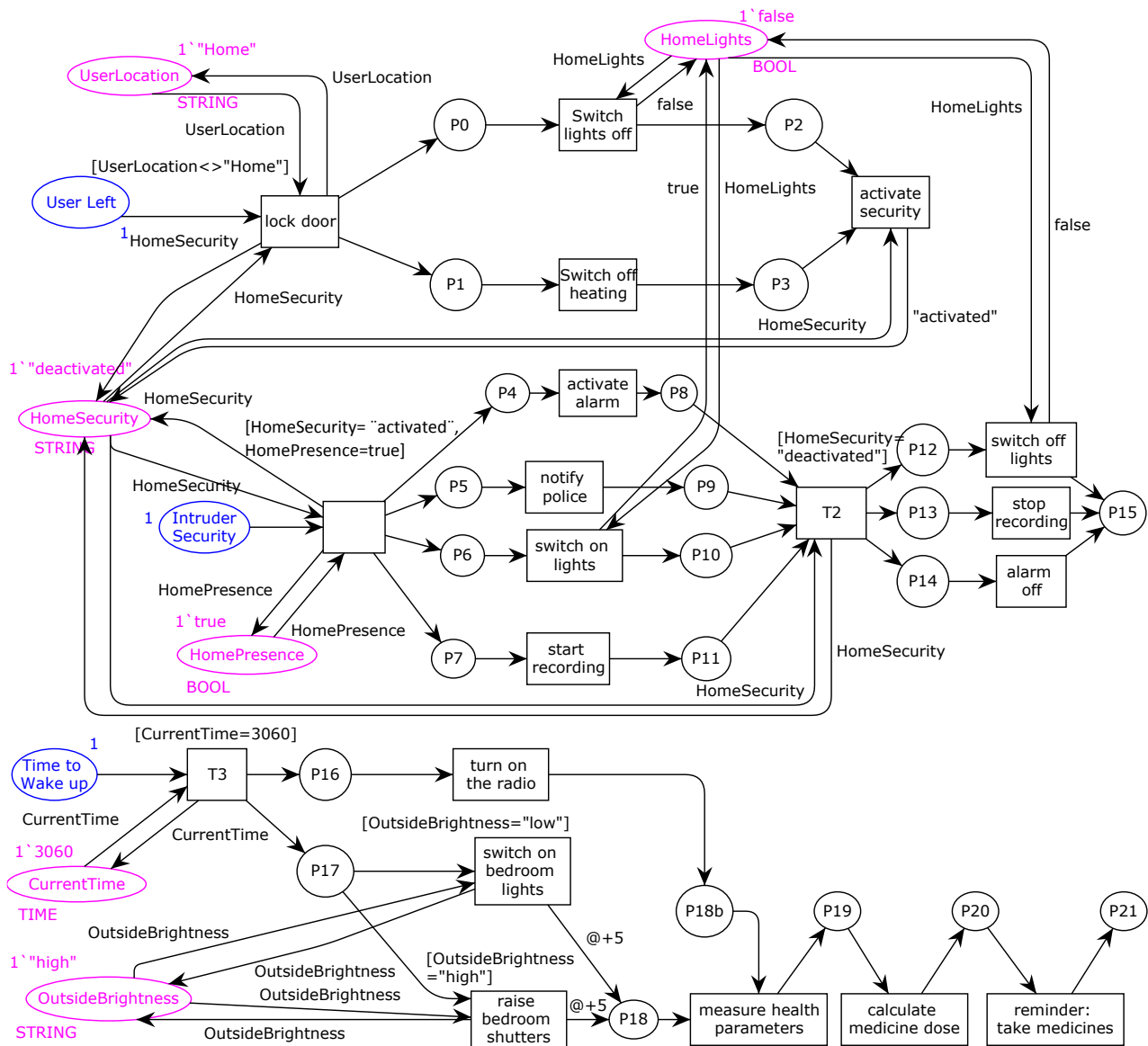


Figure 3. CP Nets that represent the "Time to Wake up", "User Left", and "Intruder security" routines of the running use case

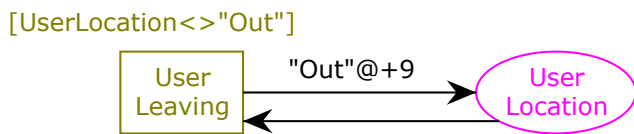


Figure 4. Simulating the user is leaving the house

the presented mappings, a transition is created and linked to each place that represents a context property whose value must change in the simulation. For instance, at the beginning of the simulation, the user location is "Home"; however, to be able to simulate the intruder routine, the user location must change. As shown in Figure , this is simulated by creating a transition "User Leaving" that changes the value of the property "UserLocation" to "Out" after certain delay.

CONCLUSION, DISCUSSION AND FUTURE WORK

In this work we have presented an approach to enable, at design time, the checking of the correctness of pervasive behaviour. Specifically, the paper proposes a set of mappings to transform the pervasive behaviour designed in task models onto CP Nets. This enables the use of simulation and checking techniques provided by CPN. For instance, by mapping the running use case to their corresponding CP Nets using the described mappings, we have been able to run several simulations to check the behaviour of the routines in different contexts. Also, we have been able to run a state space analysis to verify the correctness of the designed routines. For instance, we have validated with this analysis that the described routines do not create any loop and that all the described tasks can be executed. Also, we checked if several states were reachable, such as the state were all the appliances are off

when the user is not at home. We this checking we realized that the radio stays on when the user leaves, which is an undesired behaviour.

Thus, this work enables the creation of pervasive systems using intuitive task models, while ensuring, at the same time, a safe interaction with their users.

However, the proposed approach still presents some issues that will be addressed as future work. For instance, the current CPN notation has several limitations regarding the representation of context data. While in CATM data is stored and managed in ontologies [13], which provide a high level of expressiveness for data representation; in CPN, the data is internally represented and is strictly local with respect to any transition [2]. Therefore, to be able to use certain context data in a CP net, it needs to be passed across the net until it is used. Moreover, if any context data is instantiated in a CP net and it is needed in other CP nets, an explicit link needs to be created between the two CP nets to be able to pass the data. To deal with this problem, we have used the approach presented in [7], in which each context property is specified using a place that is connected to all the transitions where the property is required or modified (see Figure 3). However, this solution introduces more complex and bigger models that are more difficult to understand and maintain. To improve this solution, we plan to extend the presented mappings to map context-adaptive task models to the context-adaptive Petri net formalism presented in [10], which represents and manages the data using ontologies.

Furthermore, as future work we plan to implement a plugin that automates the presented mappings and, taking as input any set of CATMs, generates the equivalent context-adaptive Petri nets.

Acknowledgments

This work has been funded by the KU Leuven Research Fund (F+ Fellowship J:BOF/MS/F+/13/032) and the FWO (Fonds voor Wetenschappelijk Onderzoek) Project G0804 13N.

REFERENCES

1. J Baek Jorgensen and Claus Bossen. Executable use cases: requirements for a pervasive health care system. *Software, IEEE*, 21(2):34–41, 2004.
2. Stanislovas Bartkevičius, Ricardas Kragyns, and Kastytis Šarkauskas. Global variables in colored petri nets. *Electronics and Electrical engineering: Signal Technology*, (5):69, 2006.
3. Peter Dadam and Manfred Reichert. The adept project: a decade of research and development for robust and flexible process support. *Computer Science-Research and Development*, 23(2):81–97, 2009.
4. Kurt Jensen. *Coloured petri nets*. Springer, 1987.
5. Peter Johnson. Tasks and situations: considerations for models and design principles in human computer interaction. In *HCI International*, pages 1199–1204, 1999.
6. Seng W Loke, Sucha Smachat, Sea Ling, Maria Indrawan, et al. Formal mirror models: an approach to just-in-time reasoning for device ecologies. *Int. J. Smart Home*, 2(1):15–32, 2008.
7. Chun Ouyang, Eric Verbeek, Wil MP Van Der Aalst, Stephan Breutel, Marlon Dumas, and Arthur HM Ter Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Science of Computer Programming*, 67(2):162–198, 2007.
8. Fabio Paternò. *ConcurTaskTrees: An Engineered Approach to Model-based Design of Interactive Systems*. 2002.
9. Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.
10. Estefanía Serral, Johannes De Smedt, and Jan Vanthienen. Extending cpn tools with ontologies to support the management of context-adaptive business processes. In *Data- and Artifact- Centric BPM Workshop in BPM 2014*, 2014.
11. Estefanía Serral, Luca Sabatucci, Chiara Leonardi, Pedro Valderas, Angelo Susi, Massimo Zancanaro, and Vicente Pelechano. Incorporating users into ami system design: From requirements toward automation. In *ISD 2013*, pages 499–511. 2013.
12. Estefanía Serral, Pedro Valderas, and Vicente Pelechano. Supporting runtime system evolution to adapt to user behaviour. In *Proc. of CAiSE'10*, pages 378–392, 2010.
13. Estefanía Serral, Pedro Valderas, and Vicente Pelechano. Context-adaptive coordination of pervasive services by interpreting models during runtime. *The Computer Journal*, 56(1):87–114, 2013.
14. Andrew Shepherd. *Hierarchical Task Analysis*. Taylor & Francis, London, 2001.
15. Tony Spiteri Staines. Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. In *ECBS 2008*, pages 191–200, 2008.