Int J Softw Tools Technol Transfer (2007) 9:353–369
DOI 10.1007/s10009-007-0035-0

SPECIAL SECTION CPN 04/05

Requirements Validation: Execution of UML Models with CPN Tools

Ricardo J. Machado · Kristian Bisgaard Lassen · Sérgio Oliveira · Marco Couto · Patrícia Pinto

Published online: 13 March 2007

Abstract Requirements validation is a critical task in any engineering project. The confrontation of stakeholders with static requirements models is not enough, since stakeholders with non-computer science education are not able to discover all the inter-dependencies between the elicited requirements. Even with simple unified modelling language (UML) requirements models, it is not easy for the development team to get confidence on the stakeholders' requirements validation. This paper describes an approach, based on the construction of executable interactive prototypes, to support the validation of workflow requirements, where the system to be built must explicitly support the interaction between people within a pervasive cooperative workflow execution. A case study from a real project is used to illustrate the proposed approach.

1 Introduction

Clients (normally, stakeholders) and developers (system designers and requirements engineers) have, naturally, different points of view towards requirements. A requirement can be defined as "something that a client needs" and also, from the point of view of the system designer or the requirements engineer, as "something that must be designed". The IEEE 610 standard [1] defines a requirement as: (1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents; (3) a documented representation of a condition or capability as in (1) or (2).

Taking into account these two distinct perspectives, two different categories for requirements can be conceived:

- *User requirements* result directly from the requirements elicitation task [2], as an effort to understand the stakeholders' needs. They are, typically, described in natural language and with informal diagrams, at a relatively low level of detail. User requirements are focused in the problem domain and are the main communication medium between the stakeholders and the developers, at the analysis phase.
- *System requirements* result from the developers' effort to organize the user requirements at the solution domain. They, typically, comprise abstract models of the system [3], at a relatively high level of detail, and constitute the first system representation to be used at the beginning of the design phase.

The correct derivation of system requirements from user requirements is an important objective, because it assures that the design phase is based on the effective stakeholders' needs. Some existent techniques

This work has been supported by projects uPAIN (AdI/IDEIA/70/2004/3.1B/00364/007) and STACOS (FCT/POSI/CHS/48875/2002).

R. J. Machado (✉) · S. Oliveira · M. Couto · P. Pinto
Department of Information Systems, University of Minho, Minho, Portugal
e-mail: rmac@dsi.uminho.pt

K. B. Lassen
Department of Computer Science, University of Aarhus, Aarhus, Denmark

[4–7] can be used to support the transformation of user requirements models into system requirements models, by manipulating the corresponding specifications. This also guarantees that no misjudgement is arbitrarily introduced by the developers during the process of system requirements specification.

However, this effort of maintaining the model continuity by applying transformational techniques can prove to be worthless, if the user requirements models are not effectively validated. Typically, the confrontation of stakeholders with static requirements models is not enough, since stakeholders with non-computer science education are not able to discover all the interdependencies between the elicited requirements. Even with simple UML (http://www.uml.org) requirements models (use case diagrams and some kind of sequence diagrams) it is not easy for the development team to get confidence on the stakeholders' requirements validation.

According to [8] there are three kinds of analysis that should be accomplished before a workflow is put into production: (1) validation, to check if the workflow behaves as expected; (2) verification, to study the correctness of a workflow; (3) performance analysis, to estimate the solution conformance with throughput times, service levels, and resource utilization. This paper is solely devoted to the first kind of analysis at the process level, i.e., we are neither considering the resource dimension where resources estimation is supposed to be reached, nor the case dimension where a concrete instance of a workflow process is analysed both in its commonalities and exceptions.

This paper describes the usage of *CPN Tools* [9] in the generation of interactive animation prototypes to allow stakeholders to be confronted with executable versions of UML use case and sequence diagrams of previously elicited requirements. This approach towards user requirements validation is illustrated with a real case study where a healthcare information system (the uPAIN system) must be built to explicitly support the interaction between people within a pervasive workflow execution.

The remainder of this paper is organized as follows. Section 2 presents the case study by informally describing the purposes of the uPAIN system. In Sect. 3, the difficulties of achieving effective requirements validation based on static user requirements models are discussed. To support that discussion, some UML models of the uPAIN system are presented. In Sect. 4, we describe the construction of Coloured Petri Nets (CP-nets) for the animation of dynamic properties of UML models. Here, the relation between the adopted stereotyped UML sequence diagrams and the CP-nets is explained. Section 5 contains the global architecture of the tool environment used to generate the interactive animation prototype. Since some interoperability issues are not technologically transparent when *CPN Tools* are used together with the *BRITNeY Animation tool*, some examples of the required XML files to perform the integration are discussed. In Sect. 6, the strategies used to design the graphical user interface of the interactive animation prototype are discussed and some usability issues are referred. This section deals with the discussion of the efforts that must be made to obtain an animation artefact that effectively involves the stakeholders in the workflow requirements validation. Section 7 concludes the paper with some final remarks, mainly devoted to the synthesis of the proposed approach's limitations and of the accomplishments achieved. Future work is also briefly referred.

2 The uPAIN System

The case study considered in this paper consists of an information system (uPAIN system) whose main concern is the process of pain control of patients in a hospital, who are subjected to relatively long periods of pain during post-surgery recovery. When a surgery is concluded, the patient enters a recovery period, during which analgesics must be administered to him in order to minimize the pain that increases as the effects of the anaesthesia gradually disappear. This administration of analgesics must be controlled according to a program which depends on factors like some personal characteristics of the patient (weight, age, …) and the kind of surgery to which the patient has been submitted. The quantity of administered analgesics must be high enough to eliminate the pain, but low enough to avoid exaggerated or dangerous sedation states. This controlled analgesia is supplied to the patient by means of specialized devices called patient controlled analgesia (PCAs). PCA is a medication-dispensing unit equipped with a pump attached to an intravenous line, which is inserted into a blood vessel in the patient's hand or arm. By means of a simple push-button mechanism, the patient is allowed to self-administer doses of pain relieving medication (narcotic) on an "as need" basis. This is called a bolus request.

The motivation for the development of the uPAIN system arises from the fact that different individuals feel pain and react to it very differently. Also, although narcotic doses are predetermined as mentioned previously, there is a considerable variability of their efficiency from patient to patient. This is why anaesthesiologists are interested in monitoring several variables, in a continu-

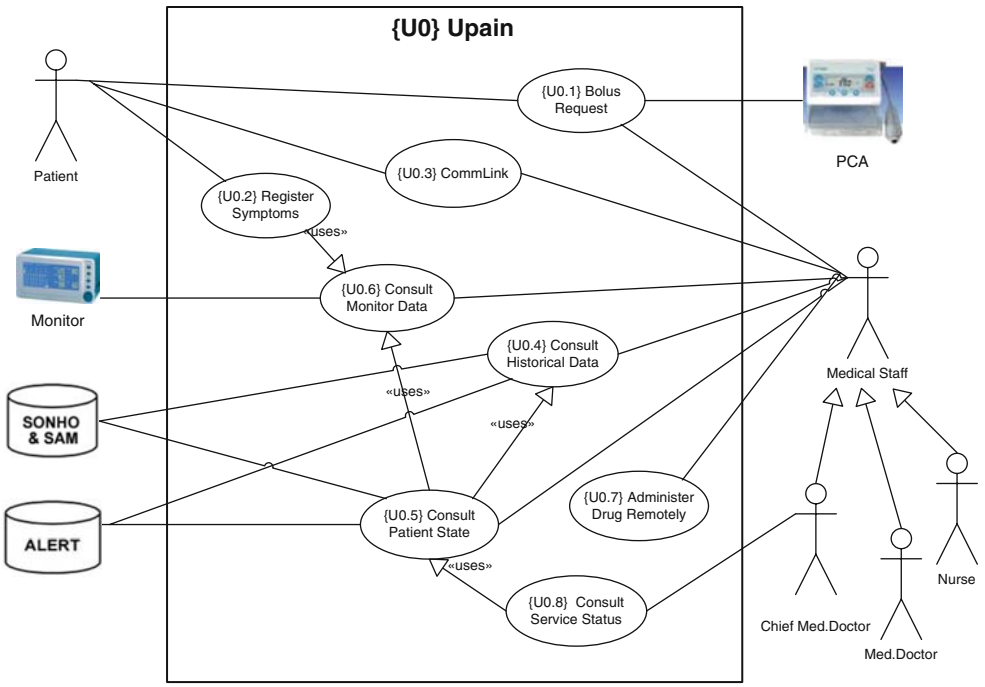

Fig. 1 Unified modelling language (UML) use case diagram for the uPAIN system

ous manner during patients' recovery, in order to increase their knowledge on what other factors, besides those already known, are relevant to pain control, and in what measure they influence the whole process. To achieve this, the main idea behind the uPAIN system is to replace the PCA push-button by an interface on a personal digital assistant (PDA), which still allows the patient to request doses from the PCA, but with the addition of the functionality of creating records in a database of all those requests, along with other data considered relevant by the medical doctors, like the values of some pre-determined physiological indicators measured by a monitor, and/or other data related to a particular patient's state, symptoms, etc. These questions may be automatically asked by the system, via the PDA, when the patient requests a dose or at regular time intervals, or even when a medical doctor decides to ask for it.

So, the uPAIN system is intended to provide a platform that enables the registration of patients' pain levels and the occurrence of several symptoms related with analgesia processes, as frequently as desired, while allowing the medical staff to be permanently aware of the occurrence of all the relevant facts of the patients' recovery and pain control processes and, simultaneously, allowing permanent remote wireless communication among system, patients and medical staff.

3 Requirements Modelling

Requirements elicitation is all about learning and understanding the needs of users and project sponsors with the ultimate aim of communicating these needs to the system developers [2]. Getting the right requirements is considered as a vital and difficult part of software development projects. Modelling and model-driven approaches provide ways of representing the existing or future processes and systems using analytical techniques with the intention of investigating their characteristics and limits [3].

UML use case diagrams are a quite adequate tool to describe user requirements at a first high-level of abstraction. These diagrams constitute a suitable means for delimiting the system boundaries, for identifying the functionalities that should be provided by the system, and for affecting external actors with specific use case functionalities. Additionally, brief textual descriptions may be provided in natural language for each use case. These diagrams are normally constructed by the developers in a tentative to document the elicited requirements. Stakeholders can read and use these diagrams to recognize the main functional areas of the system to be designed.

General functionalities of the uPAIN system are inscribed in the UML use case diagram depicted in Fig. 1.

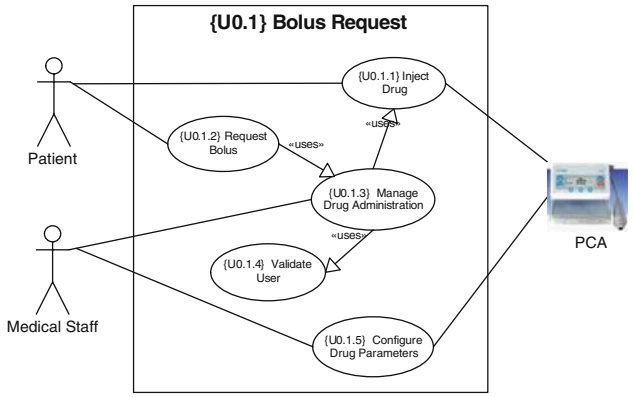

Fig. 2 UML use case diagram detailing the use case *{U0.1} Bolus Request*

A set of additional use case diagrams, as the one of Fig. 2, have been constructed to refine some of the use cases existent in Fig. 1. The corresponding textual descriptions have also been obtained.

With the exception of just a few ≪*uses*≫ and ≪*extends*≫ relationships that may already be shown between use cases, it turns out to be obvious that use case diagrams do not practically say anything about how the system should be designed, in order to supply the identified functionalities. A further step on that direction may be provided by sequence diagrams in order to illustrate the desired dynamic behaviour in what concerns its functional interaction with the environment. These diagrams are also to be constructed by the developers. Stakeholders can also read them. However, they are not comfortable with all the details these diagrams can entail.

Figure 3 depicts one UML sequence diagram for the uPAIN system that describes one macro-scenario where a patient requests a bolus. That request may be accepted by the system or originate a request for an explicit medical decision. In the later case, the doctor may decide to authorize the bolus or to reconfigure the PCA parameters.

The integration of several scenarios into only one sequence diagram (for a macro-scenario) is possible due to the new mechanisms of UML 2.0 in supporting different kinds of frames. Some more UML sequence diagrams have been constructed to capture the main system scenarios.

At the analysis phase of system development, we adopt a stereotyped version of UML sequence diagrams, where only actors and use cases are involved in the sequences, since no particular structural elements of the systems are known yet. This is illustrated by the sequence diagram of Fig. 3, whose purpose is to model the exchange of messages among the external actors and use cases depicted in Fig. 2, thus representing just a small increase in semantics detail to the use case diagram. Sequence diagrams of this kind allow a pure functional representation of behavioural interaction with the environment and are particularly appropriate to illustrate workflow user requirements.

Our stereotyped UML sequence diagrams contrast with the traditional ones that already involve system objects in the interaction with external actors, implying that those objects must be previously identified. One important issue concerning objects identification and building object diagrams is that they already model structural elements of the system, which is clearly beyond the scope of the user requirements. Additionally, the use of this kind of traditional sequence diagrams at the first stage of analysis phase (user requirements modelling and validation) require a deeper intervention of modelling skills that are hardly understandable to most stakeholders, making more difficult for them to establish a direct correspondence between what they initially stated as functional requirements and what the model already describes. So, a validation of the user requirements resulting from such an advanced model is not only more difficult to achieve, but also less trustworthy and less ensuring that the resulting system will correspond effectively to the stakeholders expectations.

4 CP-nets for Animation Prototypes

The effort to use only elements from the problem domain (external actors and use cases) in the user requirements models (use case and stereotyped sequence diagrams) and to avoid any reference to elements belonging to the solution domain (objects and methods) is not enough to obtain requirements models that are capable of being fully understandable by common stakeholders. This difficulty is mainly observable in what concerns the comprehension of the dynamic properties of the system within its interaction with the environment. This means that, even with the referred efforts, those static requirements models should not be used to directly base the validation of the elicited user requirements by the stakeholders. Instead, we use those static requirements models to derivate animation prototypes.

User-friendly visualizations of the system behaviour, automatically translated from formal systems' models specifications, accepting user interaction for validation purposes, have been generically called animations. Although it may seem a good idea, in a context where IT is offering more and more powerful multimedia capabilities, the use of animation in user requirements validation, as a means of improving the understandability of

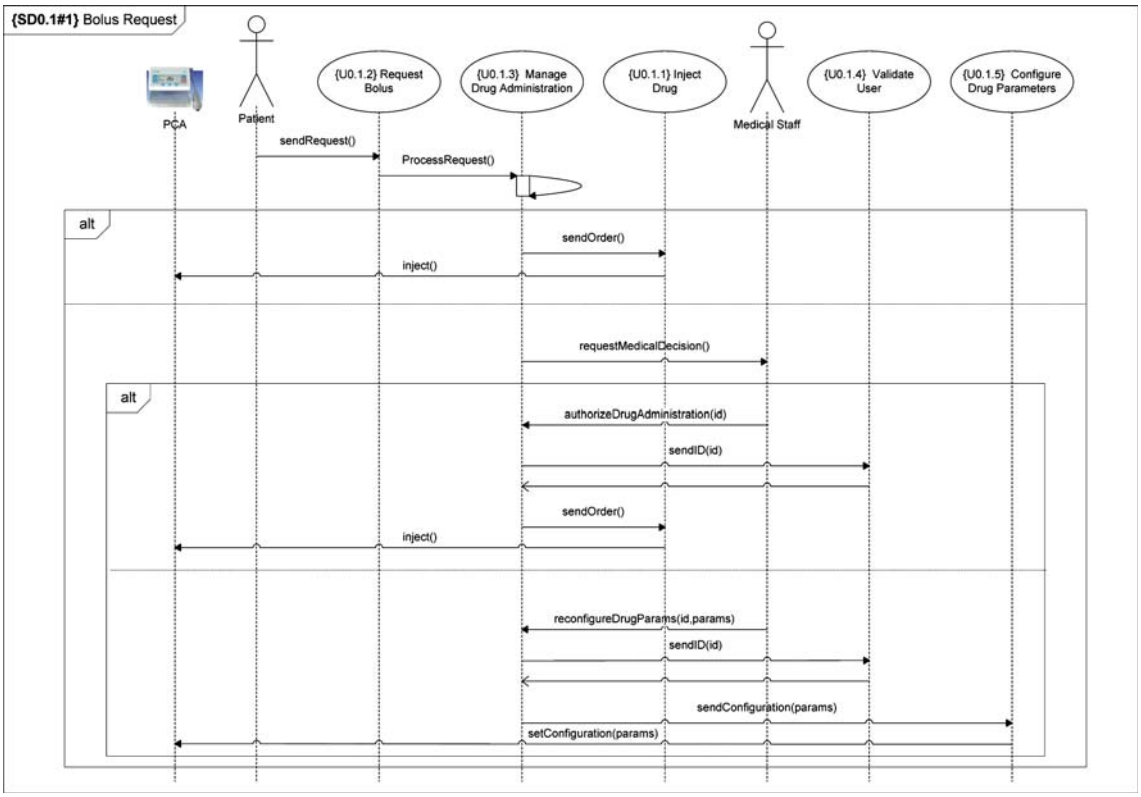

Fig. 3 UML sequence diagram of a macro-scenario for the uPAIN system

systems' models by stakeholders, has been considered by only a small number of researchers.

In [10] an empirical study has been carried out to comparatively evaluate the effectiveness of animation and narration (voice recordings of diagram explanations complemented with *PowerPoint* slides) in the process of communication of domain information to stakeholders for validation purposes, which may be seen as a sign that animation is increasingly drawing the software engineers' attention as a potentially valuable instrument for user requirements validation. The results of that empirical study were inconclusive about the effectiveness of animation, as opposed to the success of narration, but in our opinion that was due to the fact that, instead of using a meaningful user interface, the animations were of a very rudimentary type by highlighting the graphical elements of the diagrams while narration is being executed.

Some other papers have been published, reporting the use of animations to ease validation by stakeholders, as is the case in [11], where a *CORBA* API has been used to directly interpret *VDM-SL* specifications of requirements to generate a graphical user interface. Scenario-based approaches have also been used in [12] as a means of ensuring user-orientation, and also in [13], where *fluents* (boolean system states that model pairs of system actions) have been used to relate goals with scenarios and, simultaneously, support animation. In [14], virtual reality is used to support animation techniques when modelling high consequence systems (systems where errors in development have consequences of high cost).

The behaviour of the animation prototypes (proposed in this paper) results from rigorous translations of the sequence diagrams into CP-nets [15,16]. The transitions of these CP-nets present a strict one-to-one relationship with the messages in the sequence diagrams. So, for each message in a sequence diagram, one transition, in the corresponding CP-net, is created. In order to make that correspondence more evident, the name of each transition matches exactly the name of the corresponding message in the sequence diagram. Two simple rules were used for that translation: (1) Fig. 4 illustrates the rule for translating two successive messages in a sequence diagram (Fig. 4a) into a CP-net (Fig. 4b); (2) Fig. 5 illustrates the rule for translating an alternative block in a sequence diagram (Fig. 5a) into a CP-net (Fig. 5b).

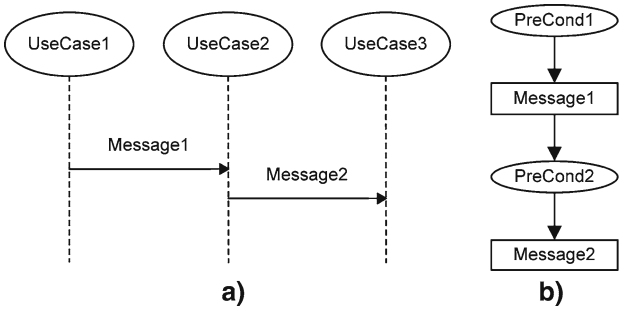

Fig. 4 Transformation of successive messages

With these transformation rules, the colour set of all of the places is the colour set *E* (or *unit*), i.e., all the tokens are the uncoloured *e* token, as can be seen in Fig. 7. This is why the arc expressions were omitted in Figs. 4 and 5.

Because all the tokens are *e* tokens, the evaluation of the expressions used in the conditions of the output arcs of a transition that has two alternative output places (e.g., transition *Message1* of Fig. 5b) cannot, obviously, depend on the colour of the tokens. Instead, the expressions in those conditions use variables that are bound by the *Output* pattern of the code segment of that transition. That situation is illustrated by the transitions *Process Request* and *Request Medical Decision* of the example CP-net of Fig. 7.

One advantage of the transformation rules suggested is that the resulting CP-nets are structurally simple and require only uncoloured tokens on the workflow paths. This would not be the case, if for the transformation rule of an alternative block of a sequence diagram (see Fig. 5), a conflict place (a place with two output arcs) was used, as depicted in Fig. 6, instead of a transition with two output arcs, as shown in Fig. 5b).

As shown in Fig. 6, if a conflict place was used to the transformation of an alternative block of a sequence diagram, the decision whether *Message2* or *Message3* should follow *Message1*, would have to be taken from the evaluation of guards associated to the transitions *Message2* and *Message3*, depending on the value (colour) of the token in the place *PreCond2*. Therefore, the colour set of the place *PreCond2* would have to be other than *E*. For that reason, a colour set *X* was used for the place *PreCond2*, and a variable *x*, of that colour set, was used for the surrounding arcs' expressions. The variable *x* would have to be bound by the *Output* pattern of the code segment of the transition *Message1*.

At an initial phase of simulations where these CP-nets are to be used, only the control of animation prototypes, for validation of the workflows by the stakeholders, is intended. For that reason, the uncoloured tokens in the workflow paths of the CP-nets is a plus, because in case of need of simulations of the CP-net model, of other types than the mere control of those animations (e.g., for performance analysis purposes), the change of the tokens' semantics, by means of the addition of colours, would be independent of the workflows' control logic, and, therefore, would not interfere with it.

4.1 Refinement Subnets

By just observing Figs. 4 and 5, it results clear that, with these transformation rules, a place of the CP-net corresponds to the interface between two consecutive messages of the sequence diagram. Therefore, a place represents a part of a use case of the stereotyped sequence diagrams, which responds to a precedent message with a subsequent message. If the refinement of a given use case is modelled with a new sequence diagram, this new sequence diagram can, in turn, be transformed into a refinement subnet. Although it is not possible to create sub-pages for places (there are no such things as "substitution places"), those output places may be

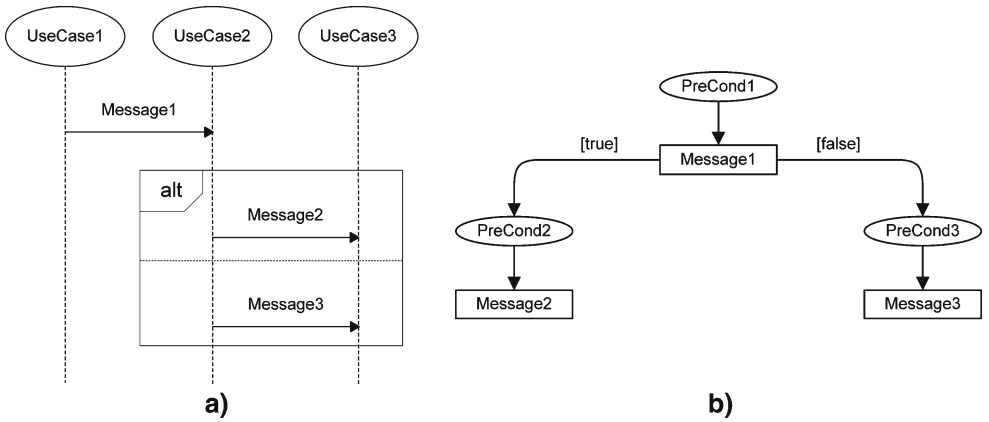

Fig. 5 Transformation of an alternative block

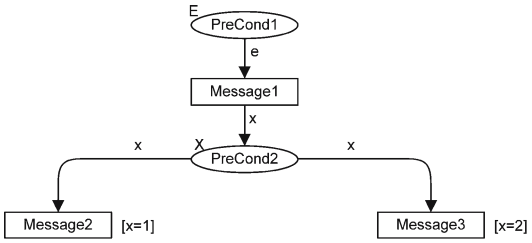

Fig. 6 Using a conflict place for the transformation of an alternative block

replaced, at the same hierarchy level where they originally appear, by refinement sub-nets (composed of one input place, one output place, and one substitution transition between them) to support the refinement of use cases. This way, the refinement subpage of each substitution transition of such refinement subnets represents the refinement of part, or the totality, of a use case.

Typically, those refinement subnets will be built after the application of the four step rule-set (4SRS) technique [7] that transforms users requirements into architectural models representing system requirements, by mapping use cases into system-level objects within a four-step approach: (1) object creation, (2) object elimination, (3) object packaging and aggregation, and (4) object association. Therefore, each transition in those subnets will correspond to the invocation of a method of a system object.

The CP-net of Fig. 7 is responsible for the animation of the use case *{U0.1} Bolus Request* (see Fig. 2) by executing the scenarios described in Fig. 3, each one corresponding to one of the three branches of the CP-net. Those nodes and arcs drawn with thinner lines were added in a later phase, and have no semantic correspondence to the sequence diagram. They were included for the purpose of tools interoperability, as explained in Sect. 5.

The CP-net represented in Fig. 8 corresponds to the top-level net of the animation prototype for the uPAIN system. Thick lines were used to represent the elements that correspond to the main animation paths. Most of the transitions of this CP-net correspond to the use cases in Fig. 1. For example, the refined CP-net of the substitution transition *bolus request* of Fig. 8 corresponds to Fig. 7.

Because the transformation rules depicted in Figs. 4 and 5 are to be applied only to sequence diagrams, direct links between the use case diagram of Fig. 1 and the CP-net of Fig. 8 were not intended to follow explicit rules. Instead, the link between the UML diagrams and the CP-nets is obtained in two steps: (1) sequence diagrams are constructed after identifying scenarios that involve use cases and actors; (2) CP-nets are derived from the sequence diagrams by applying the transformation rules.

Sequence diagrams transmit partial views for the interaction between the system and its environment, allowing the adoption of an evolutionary approach, by considering a set of sequence diagrams to have a partial evaluation of the requirements and then progress with more detailed requirements. In the uPAIN system, the animation prototype reflects only a top-level description of the system. After the validation of this top-level model, a set of additional animations, based on refined sequence diagrams at the solution level (where objects would already appear), can be constructed.

5 Tools Integration

The implementation of the interactive animation prototype demanded the usage of several technologies. The integration of tools was mainly based on XML files. Figure 9 shows the global architecture of the tool environment used to generate the animation prototype. It is composed of a model executor and an animation tool. The model executor includes a *CPN editor* and a *CPN simulator*, both from *CPN Tools*. The animation tool used corresponds to the *BRITNeY Animation tool* [17].

With *BRITNeY Animation tool* it is possible to use pre-defined plug-ins (or write our own plug-ins) for executing some animation behaviour in the model. The predefined plug-ins include *SceneBeans* [18] (an animation framework), message sequence charts (for displaying the passing of messages) and plot graphs. The writing of our own plug-ins involves the coding of Java classes and the creation of an XML description of the plug-in. *BRITNeY Animation tool* will automatically generate the code needed for the simulator to know of and use those plug-ins.

It is possible to execute behaviours in the *BRITNeY Animation tool* while simulating models in *CPN Tools*. Behaviours are executed through certain Standard Meta Language (SML) functions which in turn call the corresponding Java methods. The names of the functions correspond to those of the Java methods. When an SML function calls a Java method it simply corresponds to the logic of an RMI call. The method name and arguments are passed over to the interface of the *BRITNeY Animation tool* and the return value of the executed method is passed back from the interface. If the method *M* in class *C* has the signature *int M (int x, string y)*, then it could be invoked as *C.M (42, "Hello World")*. However, this is just an example to explain the way to

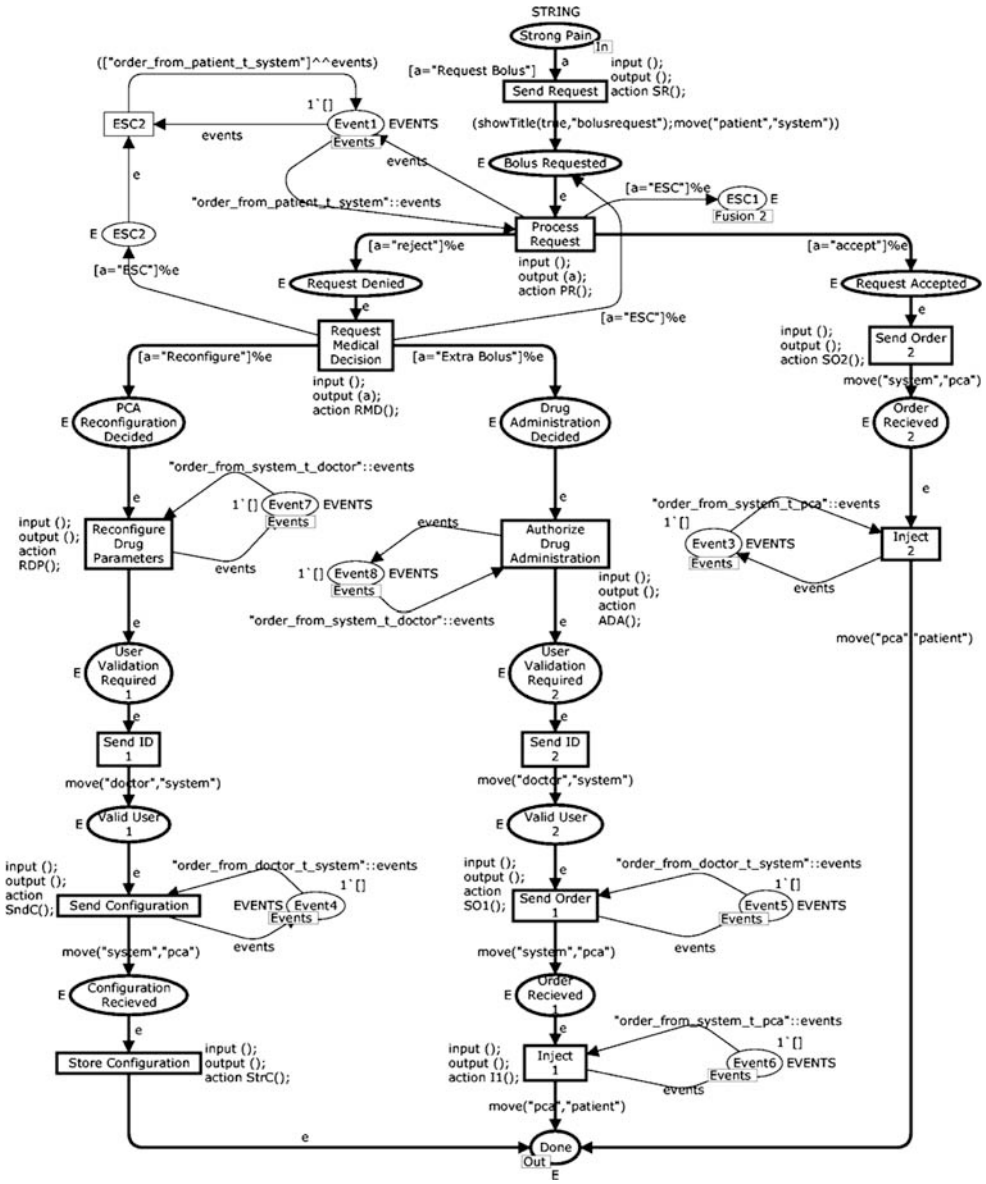

Fig. 7 CP-net responsible for the animation of the use case *{U0.1} Bolus Request*

use the Java methods in the CP-net model (see [17] for complementary explanations). These behaviours, or methods, can be executed anywhere in the CP-net model where an expression is allowed. So, it can be on an arc expression, code segments on transitions (these are specific for *CPN Tools*), and so on. This is a nice feature for debugging and for understanding the way the model affects the animation.

The *BRITNeY Animation tool* can also be executed as a standalone program, using e.g., Java WebStart to enable web browser integration. This feature is very useful to generate an autonomous animation prototype which allows stakeholders to "play with" without the interference and the presence of elements from the development team. This approach to validation was experimented with and proved to be very effective. This empowerment of the stakeholders promoted a deeper involvement of them in the analysis phase that not only assured better validation results, but also allowed the complementary elicitation of workflow requirements.

The interactive animation prototype for the uPAIN system is depicted in Fig. 10. The usage of *SceneBeans* allowed the animation of actors and message passing. *SceneBeans* provides a parser that translates XML documents into animation objects. A *SceneBeans* document is contained within a top-level <*animation*> element that contains five types of sub-elements: (1) a single <*draw*> element defines the scene graph to be rendered (e.g., the icons representing the doctor, the nurse, the patient); (2) <*define*> elements define named scene

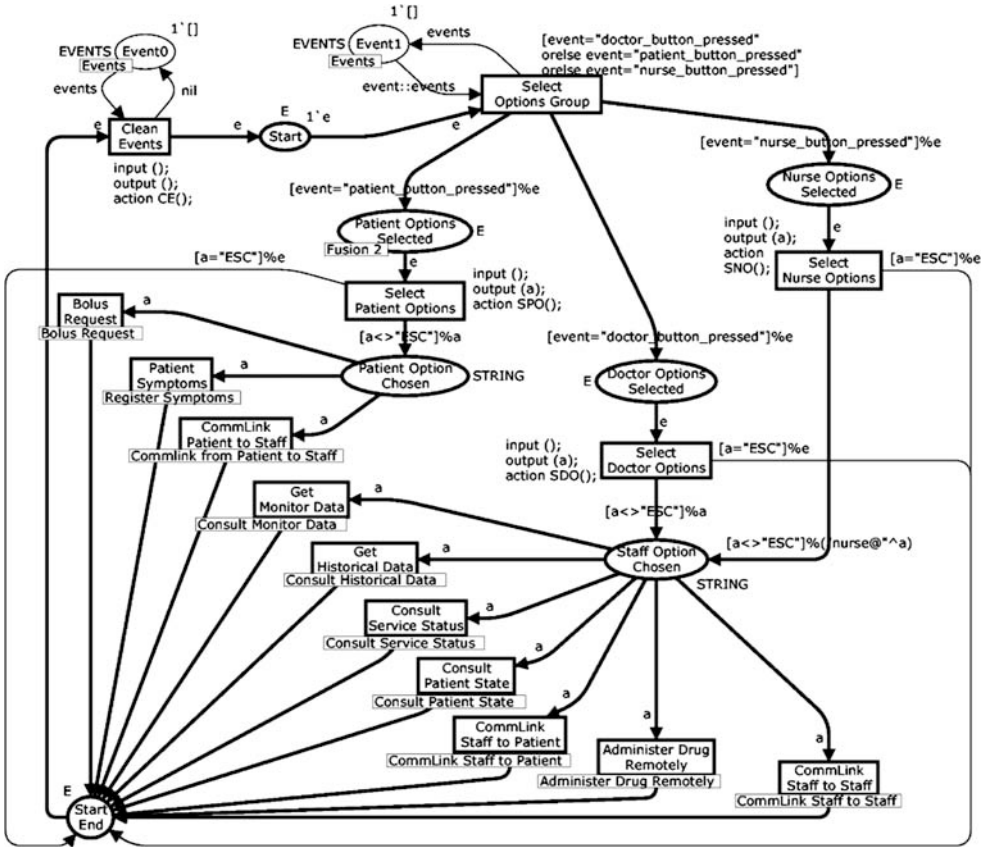

Fig. 8 Top-level CP-net of the animation prototype for the uPAIN system

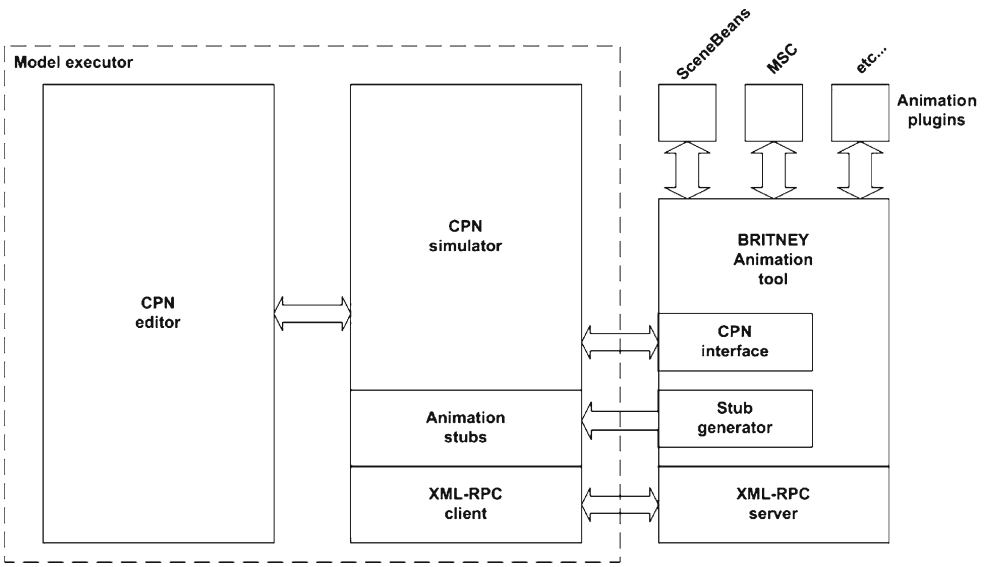

Fig. 9 Global architecture for prototype animation

graph fragments that can be linked into the visible scene graph; (3) <*behaviour*> elements define behaviours that animate the scene graph (e.g., the animation of the drug injection from PCA icon to the patient icon); (4) <*event*> elements define the actions that the animation performs in response to internal events (e.g., the cleaning of the info text at the end of the drug injection animation); (5) <*command*> elements name a command that can be invoked upon the animation and define the actions taken in response to that command (e.g., the

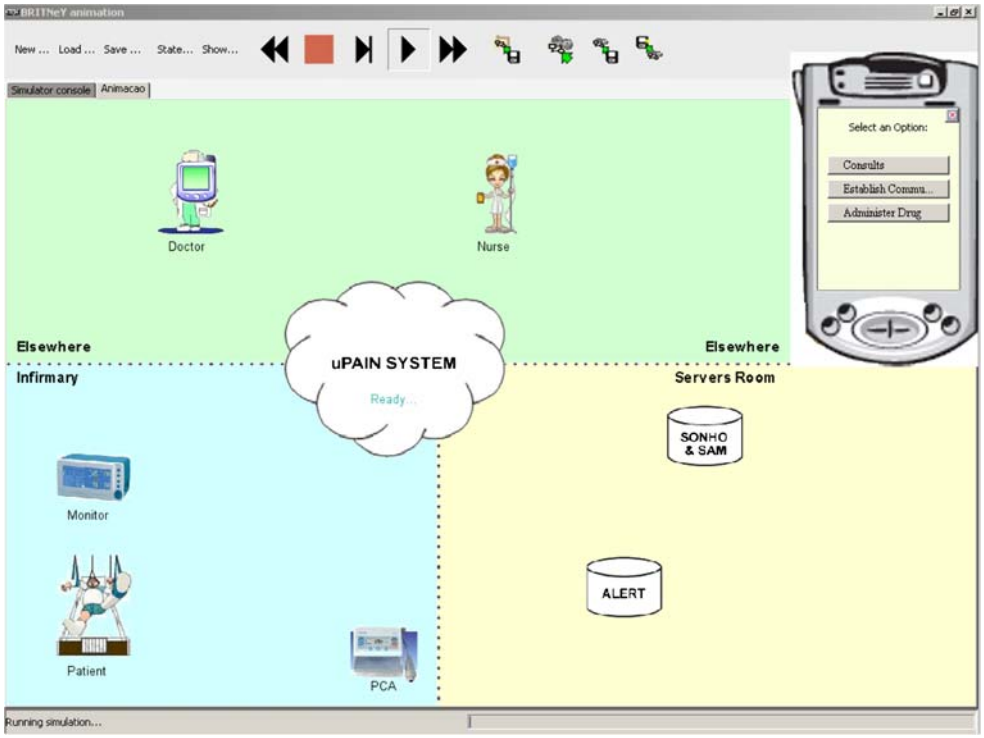

Fig. 10 Interactive animation prototype for the uPAIN system

invocation of the behaviours responsible for the drug injection animation).

Figure 11 shows an example of a code segment that was used in our <*draw*> element. This code segment is responsible for the creation of the icons for the uPAIN system cloud (the first <*primitive*> element, inside the first *translate* type <*transform*> element), the patient (the second <*primitive*> element, inside the second *translate* type <*transform*> element, which, in turn, is inside an <*input*> element, because the patient icon works as a button) and the black ball (the third <*primitive*> element, inside the third *translate* type <*transform*> element) that represents the messages between actors. To animate the ball, we used the <*animate*> element, which means that parameters *x* and *y* will be animated by the behaviours *xf_patient_to_system* and *yf_patient_to_system*, respectively.

Figure 12 shows the behaviours, the commands and the events that are responsible for moving the ball from the patient icon to the uPAIN system icon. When the simulator invokes the command *f_patient_t_system_cmd*, the <*reset*> and <*start*> elements execute the behaviours corresponding to the movement of the ball and the displaying of its textual info. When the execution of a behaviour ends, the animation will trigger the associated event, and this will start other behaviours, like *xball_out* (to hide the ball), *fadeout_info* (to hide the textual info), or *hide_patientpda_icon* (to hide the patient PDA icon). At the end, the <*annouce*> element announces the event. This last action is crucial, because it allows the *CPN simulator* to capture the event.

Communication between *SceneBeans* objects in the animation and the CP-net model can be done in two ways: (1) asynchronously, here the CP-net model simply invokes a command on a *SceneBeans* object and proceeds simulating, not caring for the moment when the animation behaviour that was executed terminates; (2) synchronously, here the CP-net model, again, invokes a command on a *SceneBeans* object, but, instead of just proceeding, the CP-net model waits for a particular event to arrive (e.g., the event "ball moved from patient to system"). This event would be broadcasted by the animation command that was executed when it terminates, to let the CP-net model know that this animation has completed. Synchronous interactions with *SceneBeans* objects must be carefully analysed; otherwise, animations that should be executed in sequence will be executed concurrently. It is necessary to determine which animation behaviours are to be completed before any other can proceed (synchronous) and those which can occur in any order (asynchronous). Invocations on *SceneBeans* objects are asynchronous in the

Fig. 11 Drawing in *SceneBeans*

```
<draw>
...
<transform type="translate">
      <param name="translation" value="(${systemX},${systemY})"/>
     <transform type="scale">
     <param name="x" value="1"/>
     <param name="y" value="1"/>
       <primitive type="sprite">
           <param name="src" value="upainimages/system.gif"/>
       </primitive>
    </transform>
 </transform>
...
<input type="mouseClick" id="patient_button">
 <param name="releasedEvent" value="patient_button_pressed"/>
 <transform type="translate">
      <param name="translation" value="(${patientX},${patientY})"/>
       <primitive type="sprite">
           <param name="src" value="upainimages/patient.gif"/>
       </primitive>
 </transform>
 </input>
<transform type="translate" id="ball">
      <param name="translation" value="(-50,-200)"/>
      <animate param="x" behaviour="xf_patient_t_system"/>
      <animate param="y" behaviour="yf_patient_t_system"/>
      ...
      <primitive type="circle">
            <param name="radius" value="10"/>
      </primitive>
 </transform>
...
</draw>
```

sense that, per default, they do not broadcast any event; this has to be specified in the *SceneBeans* XML specification.

After creating all the behaviours, commands and events, which allow the animation to announce events and receive commands from the *CPN simulator*, the next step is to create Java classes. *SceneBeans* have the limitation of not allowing the user to input dynamic contents. In fact, *SceneBeans* only allows the creation of animations, based on static behaviours, defined in an XML file. The Java classes we created are responsible for showing the graphical interfaces of the PDAs and for sending the corresponding user (of the animation prototype) inputs to the *CPN simulator*. For instance, the Log Window that shows all the messages sent between actors demanded the creation of a Java class (*Messenger*) that receives the messages from the *CPN simulator*. To add a new message to the list of messages of the Log Window, we simply invoke *Messenger.createAndShowGUI("message")*. After creating the Java classes, an XML description must be constructed so that the *BRITNeY Animation tool* recognizes them as plug-ins (see Fig. 13).

Java classes in defined animation plug-ins can be instantiated through SML functors that *BRITNeY Animation tool* generates. SML functors are "abstract" SML structures which can be instantiated. A Java object is instantiated by, e.g., *structure anim = SceneBeans(val name = "Name")*, which instantiates an object from the *SceneBeans* class. Methods on the instantiated *anim* are accessed as public methods defined in the *SceneBeans* class. Another example is the function *SPO ( )* in the transition *Select Patient Options* of Fig. 8 that contains the following code to invoke methods to our Java objects:

Messenger.cleanText ( );
Ppda.createAndShowGUI ("mainmenu");
Ppda.getValueString ( );

SceneBeans objects provide also some methods to control the animation. For instance, the calling of function *move ( )* in the CP-net of Fig. 7 consists in an

```
<!-- behaviours -->
<co id="f_patient_t_system" event="msg_f_patient_t_system" state="stopped">
<behaviour algorithm="move" id="xf_patient_t_system">
    <param name="from" value="${patientX}+40"/>
    <param name="to" value="${systemX}+60"/>
    <param name="duration" value="${ballSpeed}"/>
</behaviour>
<behaviour algorithm="move" id="yf_patient_t_system">
    <param name="from" value="${patientY}+70"/>
    <param name="to" value="${systemY}+60"/>
    <param name="duration" value="${ballSpeed}"/>
</behaviour>
</co>
<!-- commands -->
<command name="f_patient_t_system_cmd">
  <reset behaviour="f_patient_t_system"/>
  <start behaviour="f_patient_t_system"/>
  <set object="info" param="text" value="Receiving request..."/>
  <reset behaviour="fadein_info"/>
  <start behaviour="fadein_info"/>
</command>
<!-- events -->
<event object="f_patient_t_system" event="msg_f_patient_t_system">
  <reset behaviour="xball_out"/>
  <start behaviour="xball_out"/>
  <reset behaviour="fadeout_info"/>
  <start behaviour="fadeout_info"/>
  <reset behaviour="hide_patientpda_icon"/>
  <start behaviour="hide_patientpda_icon"/>
  <announce event="order_from_patient_t_system" />
</event>
```

Fig. 12 Defining behaviours, commands, and events in *SceneBeans*

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.3"
"http://jpf.sourceforge.net/plugin_0_3.dtd">
<plugin id="UPAIN" version="0.1.0">
        <requires>
                <import plugin-id="AnimationTools"/>
        </requires>
        <runtime>
                <library id="upain_jar" path="UPAIN.jar" type="code">
                        <export prefix="*"/>
                </library>
        </runtime>
        <extension plugin-id="AnimationTools" point-id="Animation" id="ScenarioSelector">
                <parameter id="class" value="views.ScenarioSelector"/>
        </extension>
        <extension plugin-id="AnimationTools" point-id="Animation" id="Ppda">
                <parameter id="class" value="views.Ppda"/>
        </extension>
        <extension plugin-id="AnimationTools" point-id="Animation" id="Dpda">
                <parameter id="class" value="views.Dpda"/>
        </extension>
        <extension plugin-id="AnimationTools" point-id="Animation" id="Npda">
                <parameter id="class" value="views.Npda"/>
        </extension>
        <extension plugin-id="AnimationTools" point-id="Animation" id="Messenger">
                <parameter id="class" value="views.Messenger"/>
        </extension>
                <extension plugin-id="AnimationTools" point-id="Animation" id="Chat">
                <parameter id="class" value="views.Chat"/>
        </extension>
</plugin>
```

Fig. 13 Defining Java classes as plug-ins of *BRITNeY Animation tool*

invocation of the method *invokeCommand* to the *SceneBeans* object *anim* (Fig. 14), which is responsible for invoking the previously defined commands in the XML file (Fig. 12). In this case, the invoked command corresponds to the movement of the black ball between the actors of the animation.

Additionally, it is possible to capture events announced by the animation. In our animation prototype

Fig. 14 Declaring and instantiating objects in *CPN Tools*

```
▶Tool box
▶Help
▶Options
▼SD0.1k.cpn
   Step: 0
   Time: 0
  ▶Options
  ▶History
  ▼Declarations
    ▶Standard declarations
    ▶Custom Declarations
    ▼Animation setup
      ▼structure anim = SceneBeans(val name = "Animacao");
      ▼structure chat = Chat(val name = "Chat")
      ▼structure scenarioselector = ScenarioSelector(val name = "SCENARIO SELECTOR");
      ▼structure messenger = Messenger(val name = "LOG WINDOW");
      ▼structure ppda = Ppda(val name = "PATIENT PDA");
      ▼structure dpda = Dpda(val name = "DOCTOR PDA");
      ▼structure npda = Npda(val name = "NURSE PDA");
    ▶fun readyRunning
    ▶fun showHabRects
    ▼fun move(from:STRING, to:STRING)=
     (anim.invokeCommand("f_"^from^"_t_"^to^"_cmd");e)
    ▶fun moveMessage
    ▶fun notify
    ▶fun showPdaIcon
    ▶fun showTitle
    ▶fun delim
```

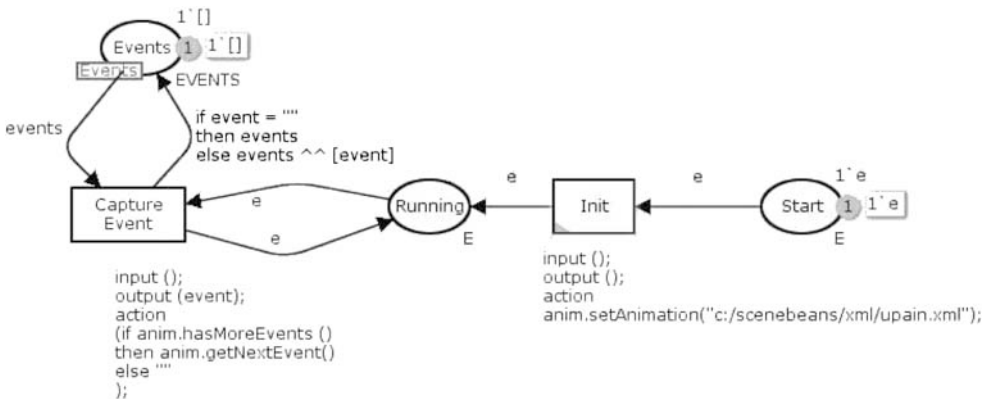

Fig. 15 *Events* CP-net subpage

we included one CP-net subpage called *Events* (Fig. 15) that is composed by two distinct parts: one is responsible for the initial loading of the XML animation description (places *Start* and *Running*, and transition *Init*); the other part includes the transition *Capture Event* (captures all the events announced by the *SceneBeans* animations and places them, in the form of a string list, in the place *Events*) and the place *Events*. This place *Events* belongs to a global fusion set of places that are connected to every transition where the capture of specific events is required (see, for instance, the nodes and arcs drawn with thinner lines in Figs. 7 and 8). All these fusion places are named *EventN* (where *N* is a digit that serves only as a distinguishing character, because *CPN Tools* do not accept places with the same name, in the same page) and have no semantic meaning from the workflows' point of view. They are only needed for tool interoperability.

6 Usability Issues

According to [19], usability is considered "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". This means that, besides all the technical efforts described in the previous two sections of this paper, the effectiveness of the implemented animation prototype to involve stakeholders in the interactive execution of the elicited sequence diagrams, complementary elicitation of workflow requirements and validation of the requirements model was also a result of a strong investment in using usability techniques in the construction of this software artefact, namely in what concerns its graphical user interface (GUI) and the comfort of exploitation of the animation prototype.

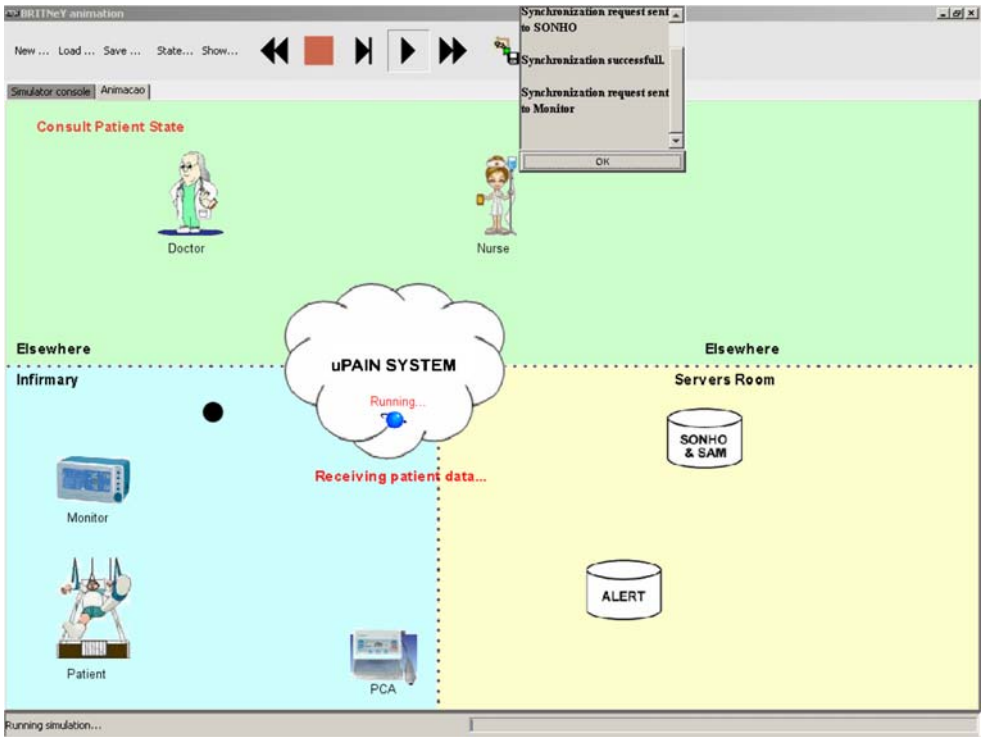

Fig. 16 Message passing in the animation prototype for the uPAIN system

The adopted GUI makes use of eight icons on the display: three proactive actors (one patient, one medical doctor and one nurse); four reactive actors (one monitor, one PCA device and two databases in use at the hospital); and the uPAIN system represented by a cloud. The adopted GUI should be obvious and intuitive to the stakeholders and thus, with the exception of the cloud and the databases, we opted for "concrete" icons. When real-world objects are represented in an icon ("concrete" icon), individuals are likely to find it more meaningful, are often familiar with the items depicted, and find it easy to make links between what is shown in the icon and the function it is supposed to represent [20]. To symbolize the uPain system (a concept which is difficult to materialize and to represent), we chose a cloud which constitutes an "abstract" icon. Forming strong systematic relations between icons and functions is very important, particularly when there are no pictorial alternatives for a given icon function [21]. To represent the uPain system we wanted an icon that emphasized its pervasive and wireless nature. Databases are also represented through an "abstract" icon which is a standard way to represent software-technology databases. We also opted for uniform icons in terms of size because we wanted to avoid stakeholders focusing on some of the icons and not others due to size differences; we wanted them to have, at the first glance, the notion of the whole GUI. On the other hand, the real-sized PDA is the biggest element and the only one that detaches from the GUI in terms of size, in order to improve the legibility of its contents.

Whenever one proactive actor is clicked with the mouse, a PDA icon appears above it and then, a real-sized version of the PDA is also displayed, showing the predefined options, corresponding to possible requests (see Fig. 10). Through each proactive actor's PDA, the stakeholder just has to select the desired option and then the corresponding sequences are executed (each one of these is formally related with one of the UML stereotyped sequence diagrams).

Each time one of the proactive actors is clicked, a black ball (representing the actor's request) is sent from the actor towards the cloud. In Fig. 16, the stakeholder interacting with the animation prototype chose the "consult patient state" option by using the PDA of the medical doctor. The snapshot in Fig. 16 corresponds to the exact moment in which the monitor is sending to the uPAIN system some physiological indicators about the patient; this data exchange is graphically represented by the black ball trajectory in the display. This snapshot also shows a log window, where all the requests and interactions are registered. At the same time, underneath the cloud, a textual expression "receiving patient

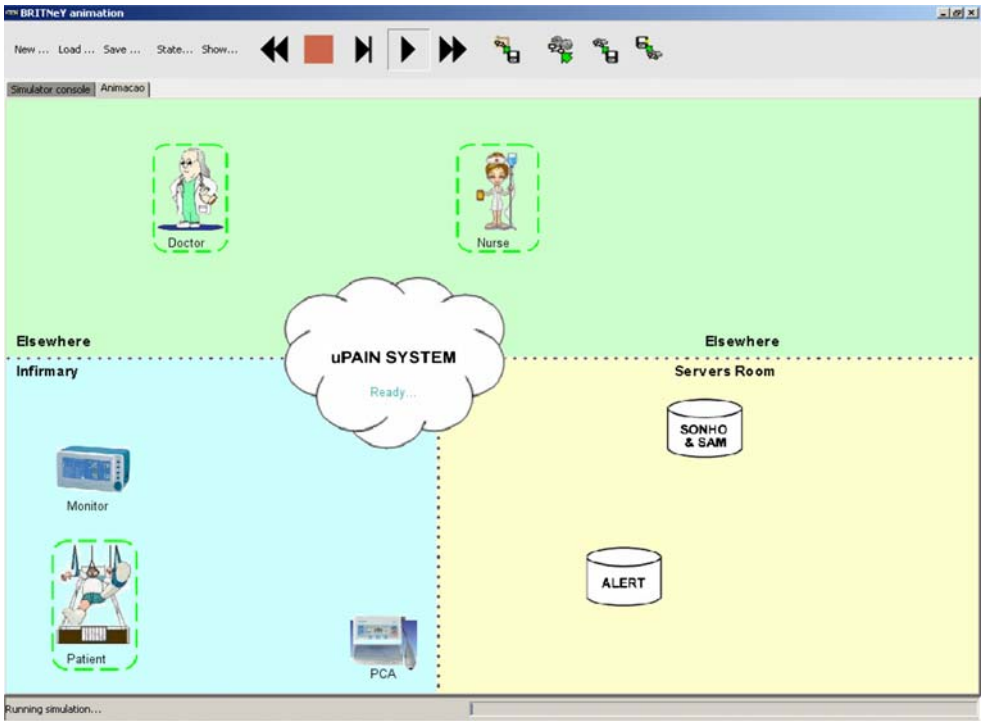

Fig. 17 Dashed line contours in the animation prototype for the uPAIN system

data" identifies the ongoing request/interaction. A caption, identifying the selected option, is displayed during the whole action in the upper left corner of the display to prevent stakeholders from forgetting the task at hand and to provide them feedback, a golden rule of GUI design suggested in [22]. It is crucial for stakeholders that the animation prototype lets them know at what point they are, at any given time in a clearly understandable way. Additionally, in Fig. 10 it is possible to observe a green coloured "Ready…" message informing that the animation prototype is ready to accept one mouse click in one of the buttons of the displayed PDA. If, in any point of the simulation, the actor "uPAIN system" is in processing state, then a spinning globe appears inside the cloud and a red-coloured "Running…" message is presented (see Fig. 15).

To assure that the purpose of any graphical entity is clearly apparent and inferred (an important cognitive dimension in GUI design to deal with expressiveness [23]), a green dashed line contour was added around each proactive actor to make clear that only these are the proactive actors on which it is possible to click to produce some kind of interaction (Fig. 17). The green-coloured "Ready…" message also appears when these green dashed line contours are displayed. We also used a dashed line and different background colours to help delimit the three main areas of the GUI and grouping actors in a logical way, according to the areas in the hospital where they may be: the patient, the monitor and the PCA are always in the infirmary; the two databases are installed in the server's room; the medical doctor and the nurse can be elsewhere due to the nature of uPain system (ubiquitous); and the uPain system is "everywhere" in the hospital and so the cloud is placed in the middle of the three dashed areas. Below each actor, and to ensure that the actor is clearly identified immediately, the respective caption was added, since good labelling can guide stakeholders through the GUI with minimal search time. We also labelled the three dashed areas. This approach in GUI design contributes for a reduced cognitive load and immediate recognition in detriment of recalling in order to let stakeholders make optimal use of their high-level cognitive abilities and save them to perform the essence of work, i.e., using the high-level cognitive capacity for the more demanding work tasks such as workflow requirements validation, which is the real aim of the animation prototype.

The reduction of short-term memory load [23] was another intended goal, once in that part of memory only few information elements (typically, five to eight) can be stored simultaneously and the decay time is short (approximately 15 s). Thus, we avoided a dense area with many elements and presented only the necessary information.

The animation prototype was first demonstrated to the stakeholders with a strong involvement of the developers to explain the main approach to its usage as a software artefact to support the early execution of functional requirements. After that, the stakeholders have been given a standalone version of the animation prototype. This usage of the animation prototype has enabled the effective validation of requirements, since stakeholders generate frequently change requests to incorporate new scenarios and to adjust others already elicited, which has definitively contributed to the rapid evolution of the requirements model maturity, prior to design phase. We believe the usability concerns we adopted in designing the whole animation prototype was determinant to the success of the uPAIN project.

7 Conclusions

Static requirements models should not be used to directly base the validation of the elicited user requirements by the stakeholders, since the effort to use only elements from the problem domain in the user requirements models and to avoid any reference to elements belonging to the solution domain is not enough to obtain requirements models that are capable of being fully understandable by common stakeholders. The stakeholders' comprehension of the dynamic properties of the system within its interaction with the environment is better assured if animation prototypes, formally deduced from the elicited static requirements models, are used.

The behaviour of the animation prototypes can be specified by using CP-nets rigorously translated from use case and stereotyped sequence diagrams. An effective execution of UML models can be achieved by using *CPN Tools* to operationally implement the interaction with the stakeholders within their efforts to validate the models of previously elicited workflow requirements. Presently, the referred transformations are executed manually, which can be considered a major drawback of the proposed approach when the system to animate is of large dimension, presenting a great number of use cases and a large amount of behavioural scenarios to transform into CP-nets.

The generation of standalone versions of the interactive animation prototypes motivates stakeholders to get a deeper involvement in the analysis phase (without the interference of the development team). Usability features of the animation prototypes must also be carefully studied and experimented, before reaching the final version of the prototype in supporting the interactive execution of the elicited sequence diagrams, complementary elicitation of workflow requirements and validation of the requirements models. *CPN Tools* and *BRITNeY Animation tool* should evolve to support better the transparent generation of this kind of standalone versions and to allow a simpler start-up of an animation.

As future work, we intend to automatically generate CP-net skeletons from workflows requirements models (use case and stereotyped sequence diagrams). Additionally, we will study the possibility of using CP-nets, constructed for specifying the behaviour of the animation prototype, to base the behavioural specification of the elements that will compose the architecture of the system within the design phase. If data-flow languages (such as LabVIEW, as described in [24,25]) are used to develop the semantic layer responsible for integrating the whole ubiquitous system (embedded and mobile devices, database accesses and a service-oriented architectural platform), the asynchronous nature of CP-nets will smooth the transition from analysis to design phases in what regards behavioural models.

References

1. IEEE 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology, 1990
2. Zowghi, D., Coulin, C.: Requirements elicitation: A survey of techniques, approaches, and tools. In: Aurum, A., Wohlim, C. (eds.) Engineering and Managing Software Requirements, pp. 19–46. Springer, Heidelberg (2005)
3. Machado, R.J., Ramos, I., Fernandes, J.M.: Specification of requirements models. In: Aurum, A., Wohlim, C. (eds.) Engineering and Managing Software Requirements, pp. 47–68. Springer, Heidelberg (2005)
4. Liang, Y.: From use cases to classes: A way of building object model with UML. Inf. Softw. Technol. **45**, 83–93 (2003)
5. Whittle, J., Kwan, R., Saboo, J.: From scenarios to code: an air traffic Control case study. Softw. Systems Model. **4**(1) 71–93
6. Krüger, I., Grosu, R., Scholz, P., Broy, M.: From MSCs to statecharts. In: Rammig, F.J. (ed.) Distributed and Parallel Embedded Systems, pp. 61–72. Kluwer Academic, Dordrecht (1999)
7. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of UML models for service-oriented software architectures. In: The 12th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2005), Greenbelt, Maryland, USA, pp. 173–182. IEEE CS Press, New York (2005)
8. van der Aalst, W.M.P.: Business process management demystified: A tutorial on models, systems and standards for workflow management. In: Desel J., Reisig W., Rosenberg G. (eds.) Lecture Notes in Computer Science, vol. 3098, pp. 1–65, Springer, Heidelberg (2004)
9. Beaudouin-Lafon, M., Mackay, W.E., Andersen, P., Janecek, P., Jensen, M., Lassen, M., Lund, K., Mortensen, K., Munck, S., Ratzer, A., Ravn, K., Christensen, S., Jensen, K.: CPN/Tools: A post-Wimp interface for editing and simulating coloured petri nets. In: The 22nd International Conference on Applications and Theory of Petri Nets (ICATPN 2001), pp. 71–80, Newcastle upon Tyne, UK (2001)

10. Gemino, A.: Empirical Comparisons of Animation and Narration in Requirements Validation. Requir. Eng. **9**, 153–168 (2003)
11. Fenkam, P., Gall, H., Jazyeri, M.: Visual requirements validation: Case study in a CORBA-supported environment. In: IEEE Joint International Conference on Requirements Engineering (RE'2002) (2002)
12. Ozcan, M.B., Parry, P.W., Morrey, I.C.: Siddiqi, J.: Requirements validation based on the visualisation of executable formal specifications. In: International Conference on Computer Software and Applications, pp. 381–386, Austria. IEEE CS Press, New York (1998)
13. Uchitel, S., Chatley, R., Kramer, J., Magee, J.: Fluent-based animation: exploiting the relation between goals and scenarios for requirements validation. In: The 12th IEEE Requirements Engineering International Conference (RE'04) (2004)
14. Winter, V., Desovski, D., Cukic, B.: Virtual environment modeling for requirements validation of high consequence systems. In: Proceedings of the IEEE International Conference on Requirements Engineering, pp. 23–30 (2001)
15. Jensen, K.: Coloured petri nets: basic concepts, analysis methods and practical use. In: Monographs in Theoretical Computer Science, vols. 1–3. Springler, Heidelberg (1992–1997)
16. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. Int. J. Softw. Tools for Technol. Transf. **2**, 98–132 (1998)
17. BRITNeY Animation tool. wiki.daimi.au.dk/tincpn
18. Pryce, N., Magee, J.: SceneBeans: a component-based animation framework for java. http://www-dse.doc.ic.ac.uk/Software/SceneBeans/
19. ISO 9241-11: Guidance on Usability, 1998
20. McDougall, S.J.P., Curry, M.B., de Bruijn, O.: Exploring the effects of icon characteristics on user performance: The role of icon Concreteness, complexity, and distinctiveness. J. Exp. Psychol. Appl. **6**(4), 291–306 (2000)
21. McDougall, S.J.P., Curry, M.B., de Bruijn, O.: The effects of visual information on users' mental models: An evaluation of pathfinder analysis as a measure of icon usability. Int. J. Cogn. Ergonom. **5**(1), 59–84 (2001)
22. Welie, M., van der Veer, G., Eliëns, A.: Breaking Down Usability. Interact 99, Edinburgh, Scotland (1999)
23. Pane, J.F.: A Programming System for Children that is Designed for Usability. PhD Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, USA, May, 2002
24. Machado, R.J., Fernandes, J.M.: Heterogeneous information systems integration: organizations and methodologies. In: Oivo M., Komi-Sirviö S. (eds.) The 4th International Conference on Product Focused Software Process Improvement (PROFES'02), Rovaniemi, Finland. Lecture Notes in Computer Science Series, vol. 2559, pp. 629–643. Springer, Heidelberg (2002)
25. Machado, R.J., Fernandes, J.M.: Integration of embedded software with corporate information systems. In: Rettberg A., Zanella M.C., Rammig F.J. (eds.) From Specification to Embedded Systems Application. IFIP Series, vol. 184, pp. 169–178. Springer, Heidelberg (2005)