Journal of Universal Computer Science, vol. 17, no. 2 (2011), 243-260
submitted: 17/9/09, accepted: 7/12/10, appeared: 28/1/11 © J.UCS

A Petri Nets based Approach to Specify Individual and Collaborative Interaction in 3D Virtual Environments

Rafael Rieder
(Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
rafael.rieder@pucrs.br)

Márcio S. Pinho
(Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
pinho@pucrs.br)

Alberto B. Raposo
(Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
abraposo@tecgraf.puc-rio.br)

Abstract: This work describes a methodology that supports the design and implementation of software modules, which represent the individual and collaborative three-dimensional interaction process phases. The presented methodology integrates three modeling approaches: Petri Nets, a collaborative manipulation model based on the combination of single user interaction techniques taxonomy, and object-oriented programming concepts. The combination of these elements allows for the description of interaction tasks, the sequence of interaction processes being controlled by Petri Nets with the codes generated automatically. By the integration of these approaches, the present work addresses not only the entire development cycle of both individual and collaborative three-dimensional interaction, but also the reuse of developed interaction blocks in new virtual environment projects.

Keywords: collaborative interaction, interaction technique specification, design process.
Categories: H.5.2, I.3.6, I.3.7, I.6.5

1 Introduction

Research on collaborative manipulation of objects in immersive virtual environments (VEs) is relevant in many areas, such as simulation and training, as well as in data exploration [Ruddle, 02]. Collaborative manipulation, or collaborative three-dimensional (3D) interaction, refers to the simultaneous manipulation of a virtual object by multiple users in a VE. In simulation and training, simultaneous manipulation of objects in VEs can be used to mimic some aspects of real-world tasks. For example, in situations like product and equipment design, assembly tasks or emergency training, even when the users are not co-located in space, collaborative manipulation may provide more realistic interaction. In data exploration, collaborative manipulation is an important tool to enhance the interaction process, by moving it from being one-sided ("I do this, while you watch") to being truly collaborative, increasing insight exchange and reducing the time for task completion.

In order to better understand a virtual reality (VR) application, especially its possible intricate interaction flow, it is very helpful to use some kind of formal

description tool like Petri Nets (PN) that can describe the system function and components. This allows not only a better understanding, but also a preliminary evaluation of each phase of the system operation, which is especially useful in collaborative 3D interaction, since the collaborative metaphor concept needs the representation of parallel activities. Moreover, a formal description facilitates the automatic generation of the core application code, from graphical representations.

Besides formal specification tools, some researchers have sought to develop taxonomies able to document and specify VEs in an abstraction level closer to the user's conception instead of the designer's or programmer's views of the application [Bowman, 99] [Bowman, 04]. These approaches split the systems into smaller parts, identifying behavior patterns and allowing to encapsulate them into classes that are able to execute some relevant functionality. This approach allows for the reuse of these classes in other projects and also allows for the combination of them to build a new interaction technique, for example.

Both the use of formalisms and taxonomies aim to better define the interaction processes, reducing the time spent during the design and implementation of VEs. Therefore, an integration of both approaches can gather the best of them: system specification according to the user's level of expertise, evaluation in early stages of the development process, and the detailing of each phase of the software development process.

This paper describes a methodology able to model and to implement software modules that represent the collaborative interaction process phases. Our methodology integrates three modeling approaches: PN formalism [Murata, 89], a collaborative manipulation model [Pinho, 08] based on the combination of Bowman's single user interaction techniques taxonomy [Bowman, 99], and object oriented programming concepts. The combination of these elements allows for the description of interaction tasks, in which the sequence of the interaction processes is controlled by PNs, and whose codes are generated automatically. By the integration of a collaborative interaction techniques taxonomy, the formalism of PN and automatic code generation, the present work addresses the entire development cycle of a collaborative three dimensional interaction.

This paper is organized as follows. Section 2 summarizes work that is related to our approach. In Section 3 we present the proposed methodology and in Section 4 we present a case study with a collaborative manipulation task. Section 5 concludes the paper.

2 Background

This section presents related work in five aspects that are related to our approach: interaction techniques specification; interaction techniques taxonomies; collaborative design; collaborative manipulation; and frameworks and tools for code generation in VR applications.

2.1 Interaction Technique Specification

Smith and Duke [Smith, 99] point out that the lack of formal descriptions during the development process of VEs inhibits the identification of similarities among different

interaction techniques, leading to the "reinvention" of existing techniques. Interaction technique specification is an important task from the perspective of both users and designers. Users require interaction techniques that allow them to complete interaction tasks in a particular application and designers like to build systems that make the required interaction possible [Smith, 99].

Hynet [Wieting, 96] is a specification methodology for interaction techniques that integrates three modeling approaches. High-level PNs represent the formal base for the specification, defining the application semantics and allowing a graphical representation for the application events (the discrete part of the application). Differential Algebraic Equations handle the continuous behavior pattern of the application, and Object Oriented Concepts allow for the enhancing of the methodology expressiveness, generating concise and compact models.

Based upon HyNet, the Flownet methodology [Willans, 01] was developed for describing dynamic behavior patterns in VEs and presents a different graphical notation that allows for the specification of both the discrete and continuous behavior patterns of the application.

The Interactive Cooperative Objects (ICO) is a formal notation devoted to the specification of interactive systems [Palanque, 97]. It borrows concepts from the object-oriented programming to describe the structural or static aspects of systems, and uses high-level Petri nets to describe their dynamic aspects. According to the authors, the specification created using ICO can be simulated, which gives the possibility to prototype and test an application before it is fully implemented.

The above specification approach provides systematic methods for interaction techniques design, test, and refining, facilitating the description of systems. However, both HyNet/ Flownet and ICO are not related to any interaction technique taxonomy and do not provide any support for the automatic generation of code, which requires a deeper knowledge of the used formalisms, both by the designers and the developers, especially in the implementation phase.

2.2 Interaction Technique Taxonomies

We can view the development process of VEs under the perspective of the interaction techniques used, with the aim of classifying them in order to better understand their components, and therefore the possibilities of software reuse in new applications.

Lindeman [Lindeman, 99], for example, demonstrates a taxonomy that divides interaction techniques according to the type of manipulation technique (direct or indirect), the system actions (discrete or continuous) and the degrees of freedom controlled by the interaction technique. This approach helps to identify the parameters involved in each interaction technique, facilitating the building of new forms of interaction.

Bowman *et al*. [Bowman, 04] present a taxonomy based on task decomposition to perform a detailed analysis of the interaction process. According to them, the separation of tasks in simpler modules allows each of them to be analyzed and tested in an independent way as a tool for evaluating the usability and effectiveness of an interaction technique in a particular context or VE.

Another advantage in the use of a task decomposition taxonomy is the possibility of reuse or combination of interaction technique components in new projects. This

characteristic allows, besides flexibility, for the conception of new techniques adequate for specific situations, such as collaboration.

Csisinko and Kaufmann [Csisinko, 07] present an approach to standardize the development of 3D user interaction techniques. The authors propose to implement the techniques directly in the driver that controls the tracking device, using Python scripts. This solution uses a variation of the Bowman's taxonomy, introducing an orthogonal property to describe the level of support provided by the driver: full, partial or not implemented in the tracking middleware.

2.3 Collaborative Design

The use of collaborative methodologies during the design of the graphical user interfaces allows the interchange of experience between different development teams, and enables the creation of well-detailed system components.

Memmel and Reiterer [Memmel, 08] provide a model-based specification method and an experimental tool that integrates models with different levels of fidelity of user-interface prototyping. Users can cooperatively work on requirement models during brainstorming sessions, interacting with project artifacts through an electronic whiteboard or using their own workspace.

Another approach, presented by Arroyo *et al* [Arroyo, 08], proposes the integration of collaborative task models into a unique design model for the development of ambient intelligence systems. According the authors, the implementation of these systems is based on a blackboard architecture, which provides a well-defined high-level interface, encapsulating abstract concepts and relationships in components that describe an expected behavior or a specific physical device.

The work of Martinéz *et al* [Martínez, 08] presents a model of interaction for collaborative VEs, which allows defining the logic of an application focusing mainly on the communication process among the objects. The proposed model, besides being based on properties of the real world communication, allows the integration of task analysis to the design of the environment. For instance, user actions are mapped as channels of communication using the Bowman's task decomposition taxonomy [Bowman, 04].

Although these proposals provide robust solutions for the specification of conceptual design models, they do not support the description of specific features in the design of 3D user interfaces, such as the representation of interaction techniques and devices commonly used in immersive virtual environments.

2.4 Collaborative 3D Interaction

The need for cooperative manipulation arises from the fact that some object manipulation tasks in VEs are difficult for a single user to perform with typical 3D interaction techniques. One example is when a user, using a ray-casting technique, has to place an object far from its current position, which can be difficult if the user does not see all the surroundings of the aimed position. Another example is the manipulation of a large object without changing to a World-In-Miniature (WIM) paradigm. In both cases, two users can perform the task more easily because they can

both advise each other while performing cooperative, synchronized movements they are not able to perform alone.

In most of the known collaborative virtual environments, the simultaneous manipulation of the same object by multiple users is avoided. True collaborative manipulation has been the focus of a few research efforts. Most of these efforts used force feedback devices so that each user senses the actions of the other [Basdogan, 00] [Sallnäs, 02].

Margery *et al.* [Margery, 99] present an architecture to allow cooperative manipulation without the use of force feedback devices. The system is restricted to a non-immersive environment, and the commands that can be applied to objects are vectors defining direction, orientation, intensity and the point of application of a force upon the object. Thus, Margery's work is based on the simulation of real-world cooperative manipulation.

Earlier research by Ruddle *et al*. [Ruddle, 02] presented the concept of rules of interaction to support symmetric and asymmetric manipulation. In a subsequent work [Ruddle, 03], the same authors separated collaborative tasks into two levels of control. The high level control activities correspond to those tasks that require attention, planning and mental effort by the users to be executed. The low level control activities are quasi-autonomous activities that, once learned, are quickly executed by the users with no conscious control.

Duval *et al*. [Duval, 06] presented a cooperative manipulation technique based on "crushing points", considering the size and the geometry of the object. Two crushing points define a "skewer" across the object. According to the authors, the users feel like they are pulling the object by a virtual cord. The proposed technique uses only the user's hand position to apply translations and orientation changes to the object. The only problem reported for this technique is that rotation around the axis of the skewer is not allowed. To do so, the users have to release the object and select new crushing points, or new controls (like buttons or six degrees of freedom trackers) must be added to the interaction process.

The work of Pinho *et al*. [Pinho, 08] presents the concept of collaborative metaphor for simultaneous interaction in VEs. This metaphor is composed by a set of rules that defines how to combine each step of the interaction process, allowing that interaction techniques normally used in individual interaction be combined to compose a collaborative technique. The steps of the interaction process used are those steps defined by Bowman's task decomposition taxonomy [Bowman, 04]. The combination of the steps of interaction techniques is obtained by the distribution of the degrees of freedom of the objects' control among the users.

Riege *et al.* [Riege, 06] present a collaborative pointing technique for co-located multi-user interaction in VEs called "The Bent Pick Ray", based on the ray casting metaphor. This approach allows users to select and manipulate objects, collaboratively or not, without locking objects and preserving the visual feedback. Multiple selections and concurrent manipulations are controlled through functions that merge the inputs from multiple users. In collaborative tasks, the users are continuously informed about their connection to the object through bent pick rays, which also provides a direct feedback from the input merging process.

We use the latter two collaborative 3D interaction methodologies as case studies for the PN-based approach proposed in this work.

2.5 Tools for Code Generation

VR Frameworks try to separate functions in modules, allowing for the abstraction of the complexities of some system actions, and furthermore, the reuse of these software modules, coded by design languages.

The Unit framework [Olwal, 04] inserts an abstraction layer between the applications and its devices, and inserts application units into a data flow. Similarly, Figueroa *et al* [Figueroa, 02] propose an architecture based on pipes and filters, where information sources, such as physical devices, generate a flow of data that are propagated through interconnected filters. However, the code generation process offered by both frameworks results in interpreted code, which may compromise the quality of the interaction if the hardware does not support the application demands.

Vitzhum [Vitzthum, 06] presents a visual design language that focuses on the formal specification and supports the use of model-driven implementation. Although this approach is task-focused domain and generates less code than the traditional concepts, it is essential a previous experience with software engineering principles.

From a different perspective, Ying and Gračanin [Ying, 04] aim to understand the interaction process of existing VR applications. Analyzing an existing code, the information related to the user interaction is extracted and organized in an XML file that serves as a base for building a PN model representing the target application. By simulating the PN, one can "view" the interaction process through the PN behavior pattern, while the user is interacting with the VR application. Nevertheless, this approach is restricted to the test phase and does not contemplate previous steps of the software development process, because reverse code engineering is used to create description files.

3 The Proposed Methodology

From the literature review, it is possible to conclude that the approaches presented above have specific advantages and goals. However, in general, they do not address the entire computer application development cycle, especially concerning the final phases of debugging and code generation. Towards this goal, we developed a methodology for hierarchical development of a VR collaborative interaction process using Petri Nets, beginning from the design stage, based upon Bowman's interaction taxonomy, up to the implementation phase, relying on the object oriented programming paradigm.

The main goal of the proposed methodology is to model and implement modules that represent the steps of the interaction process. The use of formalism in conjunction with an interaction taxonomy allows for the detailed specification of the system, as well as facilitating the structuring and implementation process, encapsulating functionalities. These characteristics enable the generated modules to be tested in advance and reused later, simplifying and accelerating the development of VEs.

The graphical representation adopted here is based on Colored Petri Nets (CPNs) [Jensen, 97] because, during the modeling process, we need an easy way to differentiate the various types of data that are manipulated in a VR application. For simplification, this work refers to CPNs simply by the expression Petri Nets (PN).

The methodology presented in this paper aims to approximate the user's conception of the application from the designer and developer's point of view, modeling the application under the perspective of tasks which the user has or wishes to perform inside the VE. These tasks can be decomposed in elementary tasks that can be easily identified in most VR applications [Bowman, 99]. These elementary tasks split the interaction process into three phases: *selection*, *manipulation* and *release*. Our work adapts this taxonomy dividing the selection phase into *selection* and *attachment*. The former represents the indication of the object which the user wishes to manipulate, while the latter deals with the confirmation of this selection. Both provide feedback to the user in order to confirm their execution.

The designer needs to follow three steps to apply the methodology: 1) identify the VE tasks, according to Bowman's taxonomy, as well as the main states reached by the application after executing each task; 2) define a PN with the tasks and states identified in the previous step; 3) implement the model, using a set of classes specially developed to build the PN and to control its execution. Each of the above phases is detailed below.

Considering PNs, our methodology assigns to each basic element used in the PNs (places, transitions, arcs and tokens) a specific role or function during the interaction process in a VR application. Places define the current application state, transitions are elements that perform actions to modify the application's behavior pattern, arcs define the execution sequence and tokens are the resources available for executing the VE.

In order to illustrate the use of our methodology in the specification process of a VE, we built a virtual rotary engine application (Figure 1) in which the user's primary goal is to assembly the engine, connecting its parts. In this section we present a single user interaction, and in the following section we extend it to collaborative interaction.

3.1 Identifying Interaction and Building the PN Model

The first phase of our methodology identifies the application phases based on Bowman's taxonomy. The application starts in the *Selection State* (Figure 2, on the left) in which the user can move the pointer, looking for an object to select. From this point the *Selection Task* tests whether there is a virtual object indicated by pointer. If so, the *Selection Task* transition is fired, and the *Attachment State* is established.

At this point if the user presses and holds the selection button, the *Attachment Task* is fired attaching the selected object to the pointer and establishing the *Manipulation State*. Once this state is established the PN fires the *Manipulation Task* which allows the user to relocate the object using the pointer. If the user releases the selection button, the *Release State* place enables the firing of *Release Task*, separating the pointer from the previous selected object.

After identifying the application tasks in a high abstraction level, it is necessary to perform a task subdivision process, splitting them into smaller parts (see Table 1), based upon the operations each of them has to execute.

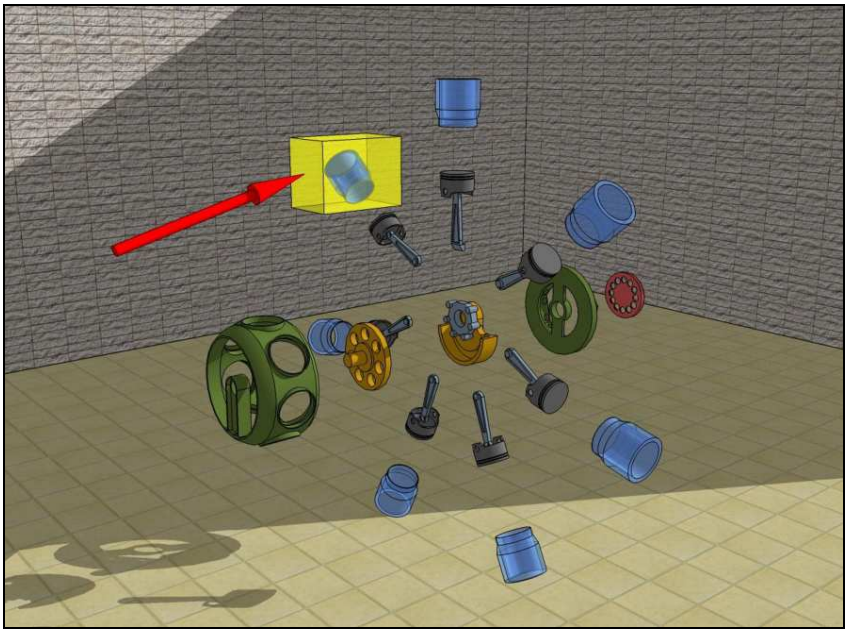

Figure 1: The virtual rotary engine application.

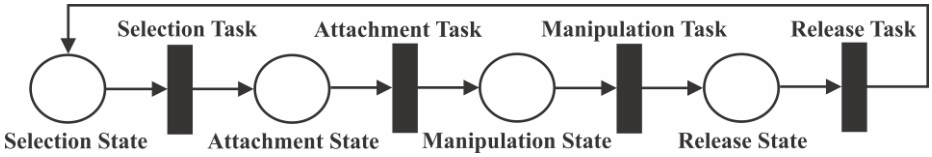

Figure 2: A high-level PN for the application.

High-Level Tasks	Basic Operations
Selection	Indication Subtask Indication Feedback Subtask
Attachment	Confirmation Subtask Confirmation Feedback Subtask
Manipulation	Positioning Subtask Repositioning Subtask
Release	Detachment Subtask Detachment Feedback Subtask

Table 1: Detailing the high-level tasks.

Following the methodology, the identification of the necessary resources (data) for each interaction phase should be initiated, which is represented as tokens in the PN. For this, PN arcs should be labeled with the token types, graphically represented by icons.

The places *Selection State*, *Attachment State*, *Manipulation State*, and *Release State* need to be constantly updated with information about the devices and control variables from the application. Therefore, tokens with these data must be inserted into them, as can be seen in Figure 3 that presents the complete PN model for the application.

Utilizing formalism, interaction devices and the application can be represented as source transitions in the PN model, as they don't have input places, being always enabled to fire and to produce tokens to the net (in this case, information about the user physical interaction and the VE state). In Figure 3 the devices are represented by triangles, while the application is represented by hexagons. These shapes are merely illustrative and serve only to help understanding the network.

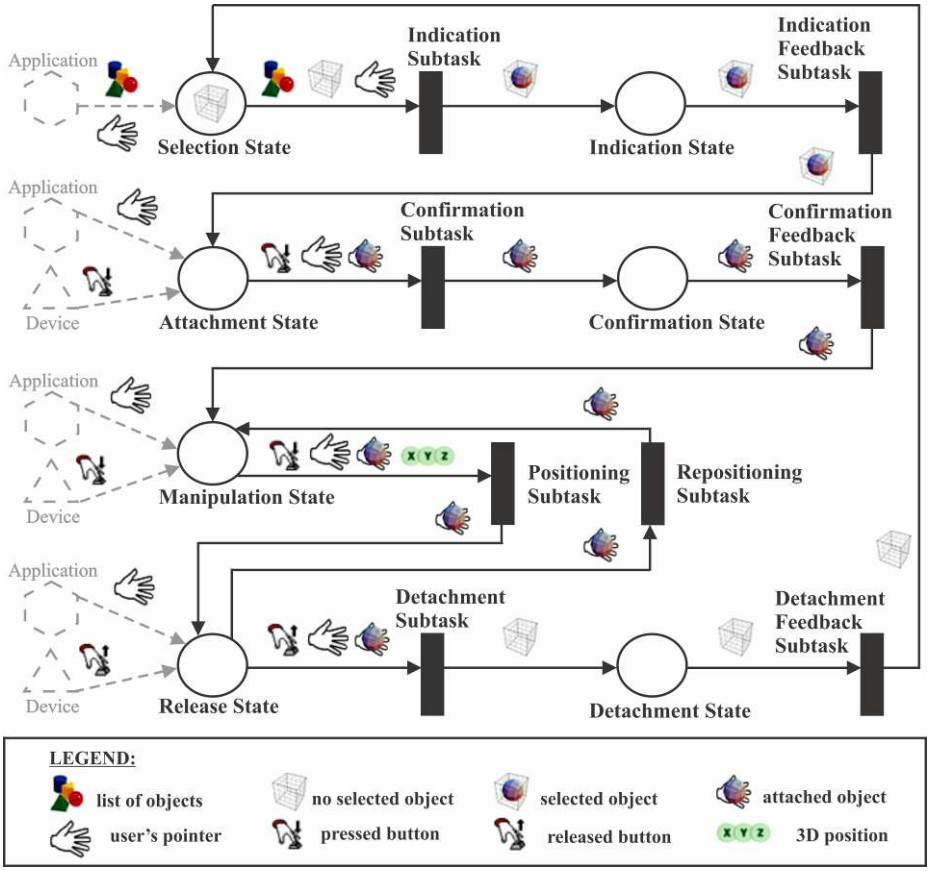

Figure 3: PN model and data resources for the modeled interaction process.

The pace of the PN simulation is controlled by the application, which tests each transition every time the user's view needs to be modified, guiding the execution of actions. In this step, the simulator fires the transitions that have their pre-conditions fulfilled. A transition's firing generates a call to a function previously assigned to the task. In other words, all the transitions are checked on every rendering cycle and fired or not, depending on the existence of the pre conditions (the necessary tokens). Therefore, it is possible to analyze the process logic together with the system and devices behavior patterns.

A complete PN cycle, presented at Figure 3 and representing the virtual rotary engine application, can be interpreted as follows: tokens are sent to the *Selection*, *Attachment*, *Manipulation* and *Release states*. When the *Indication Subtask transition* receives all the necessary tokens, a function is fired, unpacking its data and testing whether some engine part is being pointed by the user. In positive case, the transition creates a new token to represent this situation and passes it to the *Indication state*. After this, the *Indication Feedback Subtask transition* is fired, calling a function that highlights the pointed engine part.

Immediately, the *Attachment state* accepts a token that represents the selected object, and waits for the token that indicates a button being pressed by the user. When this happens, the *Confirmation Subtask transition* is fired, calling a function that attaches the engine part to the user's pointer. *Confirmation state* receives this information, which allows the firing of *Confirmation Feedback Subtask transition*, responsible for communicating the success of the attachment process using an alert sound.

A token encapsulating the attached object is sent to the *Manipulation state*, which defines the start of the manipulation process. *Positioning Subtask transition* is fired, requesting a function to update the object's position, according to the tracker data. This transition generates a new token, and sends it to the *Release state* as well as back to the *Manipulation state*, through the *Repositioning Subtask transition*. While the selection button remains pressed, the *Positioning Subtask transition* is repeatedly fired, allowing the user to freely move the engine part around the VE.

If the button is released, the *Manipulation state* will no longer fire *Positioning Subtask transition*. Concurrently, the *Release state* receives tokens informing the user's action and the manipulated engine part. *Detachment Subtask transition* is fired, releasing the object in its new VE position. Immediately, the *Detachment state* receives a token that fires the *Detachment Feedback Subtask transition*, which communicates the success of the detachment process. A token is then sent to the *Selection state*, allowing for a new engine part selection to be initiated.

3.2 Implementation Phase

In order to derive the implementation, we start with the graphical description of the PN, created in Dia editor [Dia, 10]. From this diagram, an XML specification is obtained which, in turn, originates a C++ code.

Using Dia, it is possible to add support for new types of diagrams by writing simple XML files, thereby creating specific libraries with elementary objects called "shapes". This feature also allows diagrams to be stored in XML files, facilitating the conversion of models to other codification forms, such as Java and C++ languages, or other markup languages, such as PNML (Petri Net Markup

Language) [Billington, 03]. The flexibility of the conversions allows for designers the reuse of models in new projects or different development platforms, and the use of other tools able to validate the syntactical structure of PNs, since Dia has no native support to formal validation.

In order to support the PN modeling, new shapes have been created to represent the PN elements place, transition, arc and token, organized in a library called Petri Net Interaction Process Diagram. This library, besides facilitating diagram drawing, also enables the generation of C++ code from a PN model. Figure 4 presents this library incorporated in the Dia environment.

Dia uses an XSLT (eXtensible Stylesheet Language Transformation) file format that allows conversion of XML tags to construction of a programming language, such as C++. We created two XSLT files that facilitate this conversion. The first file defines rules to generate an XML file from the graphical diagram, whereas the second contains rules to convert from the PN elements to C++ classes. These classes run the modeled PN and may be connected to a VE, coordinating its interaction.

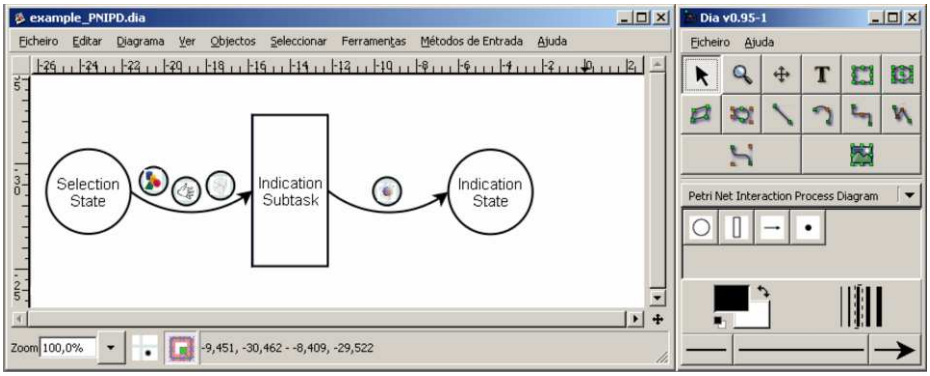

Figure 4: PN Interaction Process Diagram Sheet loaded inside the Dia environment.

4 Collaborative Manipulation Case Studies

We adapted the model of the virtual rotary engine application in order to support collaborative manipulation tasks. Figure 5 presents the new model, including two new tokens responsible for the objects' translation and rotation tasks. This way, for instance, during the manipulation step an user may control the positioning of an object in the X-Y plane, while another user may control its orientation in the Y axis. This first case study is based on the Pinho's methodology [Pinho, 08].

Each user interacting in the application has his own tokens, since the PN represents the behavior of the system as a whole. In Figure 6, for example, tokens representing each user have a specific border color (thick blue and thin red), while the shared token, representing the object has a different border style (dotted magenta). In this example, the thick blue user is responsible for the object's translation in the 3D space, while the thin red user is responsible for the rotation of the same object. For this example, we considered the arcs label definition presented in Figure 5.

In this case study, the *Manipulation state* receives the shared object, the tasks that each user may execute, and continuous information about the devices and the application. The existence of the tokens enables the PN execution in parallel, determining which task the user may perform (positioning or orienting the engine parts).

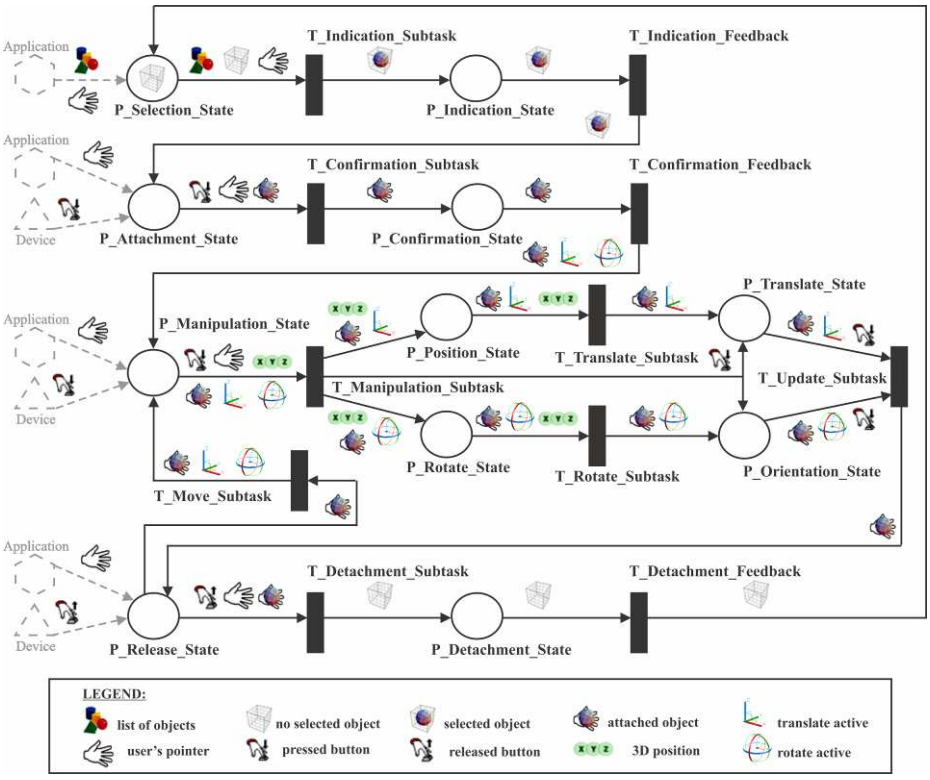

Figure 5: PN with parallel activities. "Rotation active" and "Translation active" tokens define the task to be executed.

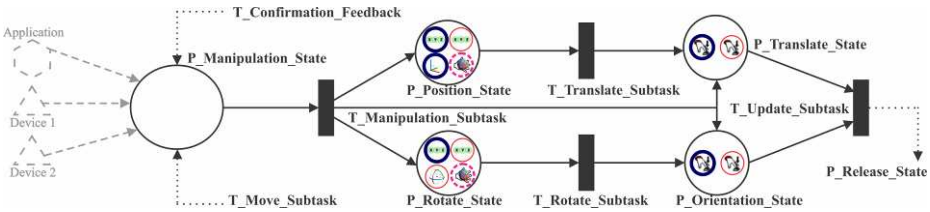

Figure 6: PN state during the cooperative manipulation.

Figure 6 still illustrates, in places *Position State* and *Orientation State*, the distribution of these activities between the users. Note that the tokens responsible for the user's decision to continue or not with the object manipulation (pressed button) are forwarded directly to places *Translate State* and *Rotate State*, as a means to wait for the conclusion of the translation and rotation tasks. When both users complete the concurrent activities, the interaction process is enabling to continue as before.

It is possible to show that the net also provides support for the representation of individual interaction. Figure 7 presents interactions of two users with different objects, in distinct steps of the interaction process (selection and attachment).

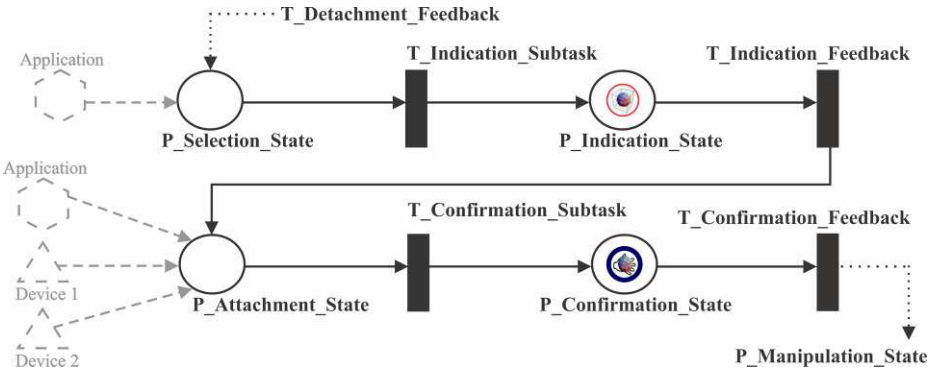

Figure 7: PN state representing the interaction with two distinct objects, in different tasks.

In order to validate our methodology with tasks and techniques applied in realistic settings, we present a second case study that illustrates our approach. In doing so, we model a bent pick ray [Riege, 06], a collaborative pointing technique discussed in Section 2.4. Part of the specification could be seen in Figure 8, which highlights the functioning of the technique during manipulation tasks.

According to the description of technique, the object could be moved simultaneously by two or more users. Techniques for merging the users' input are used, which weight the influence of interaction according to the amount of hand movement a user does. Figure 8 also shows these features, encapsulated in the transitions *Offsets Subtask*, *Weights Subtask* and *Merging Subtask*, which represent the required functions to combine the users' movement, determining a new position for the object.

This way, a user may perform the translation of an shared object in the X-Y plane, while another user perform the translation in the X-Z plane, at the same time. The merging process is started after this. Figure 9 shows this situation, considering the arcs label definition presented in Figure 8.

Rotation tasks also may occur in parallel with the translation activities, as well as specific technique steps. Figure 10 complements the previous situation, showing that orientation actions could be performed at the same time that the visual feedback of the technique, when the bending of the pick rays is defined.

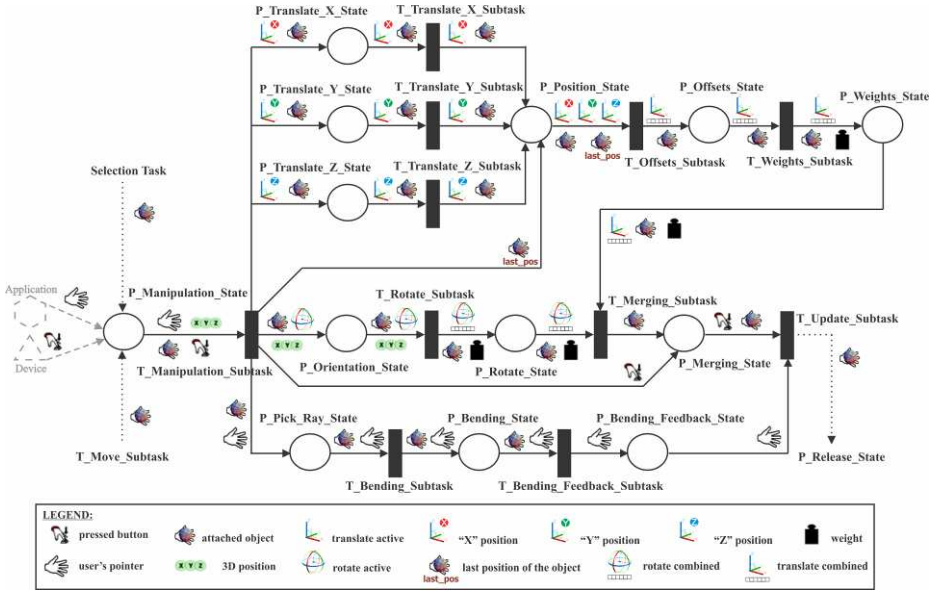

Figure 8: PN model representing the Bent Pick Ray interaction technique during the manipulation task.

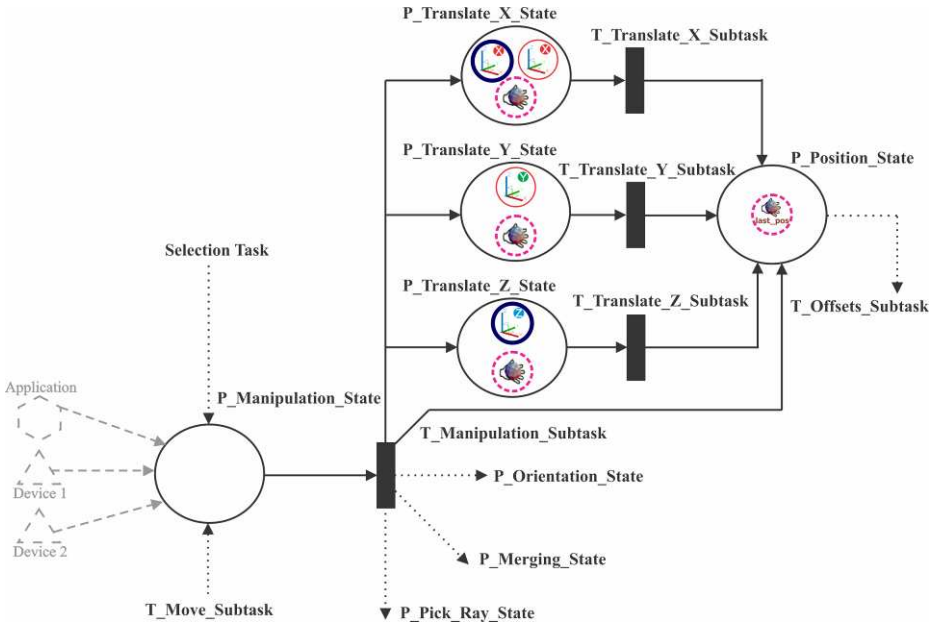

Figure 9: PN state during the simultaneous manipulation.

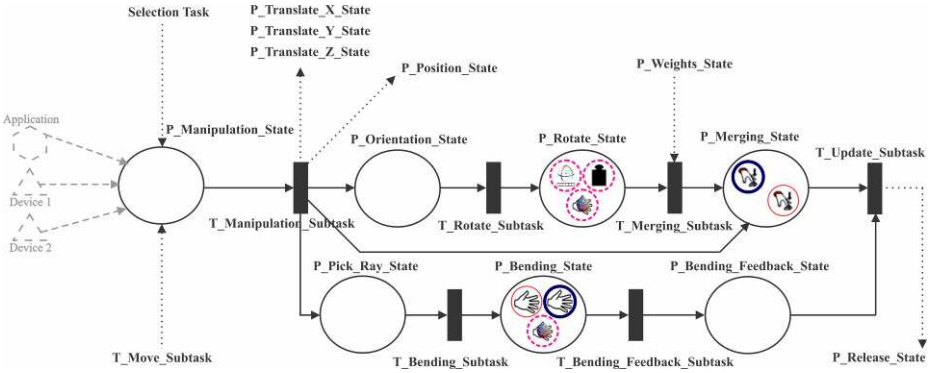

Figure 10: Parallel activities, such as interaction tasks and technique steps, could be represented using our methodology.

4.1 Discussion

With regard to the case studies, compared with the standard approach to deal with collaborative 3D manipulation, our methodology approximates the application modeling to the user's conception, allowing effective communication between designers, developers and end-users during the entire development cycle. The PN interaction process representation also facilitates the identification of the parts of the system that could be parallel or support similar situations, such as individual and collaborative interaction. This can result, for example, in simplification of the system and reduction of the design-time and development-time, since our methodology is based on the top-down development approach, which allows breaking down a system to gain insight into compositional sub-systems. So, it is possible to create interaction technique components from generic parts, and reuse them in future projects.

For this reason, our methodology also supports the interaction technique representation at different levels of abstraction, in design time, allowing basic elements to be used in new projects. The PN decomposition technique, combined with the code generation process, allows for the interaction technique features to be elucidated to the developers, who can dedicate their time and attention to improve them and optimize their codes. Moreover, the use of levels of abstraction in projects can also hide technical features from end-users, facilitating the understanding of the interaction technique during their interaction in VEs.

On the other hand, our methodology requires designers to know the basic PN concepts and rules in order to draw and revise their models, since Dia has no native support to PN projects. Depending on the system's complexity, or the development team's knowledge level, a training step may be required to prepare the professionals to design and review the PN model.

5 Conclusions

This work has presented a methodology to specify interaction tasks for VR applications using the Petri Net formalism as a base for the software design. Our

methodology aims to facilitate the application development from the conception and design phases to the implementation, test and documentation processes. The option of code generation from the graphical model helps the communication between designers and developers, avoiding the breaking of paradigms, and speeding up the development process. Moreover, an application built with our methodology can offer optimized functions, allowing users to complete their interaction tasks in an intuitive way.

Even though we have adopted the C++ language for the code generation process as described in Section 3.2 (Implementation phase), there is no restriction on the use of another programming language as default for this process. Our approach has adopted the C++ language because the majority of VR applications is developed in this language or uses resources from graphics/scene libraries written in it. However, if the designer needs to export PN models (stored in XML files) to another language, such as Java, he must create an XSLT file containing rules to convert the PN elements into Java classes.

Since the control of the PN simulation stages is autonomous, it would be interesting to show the application running process in a graphical animation superimposed over the PN graph itself, parallel with the application usage. Currently we only generate a textual output during the application's execution. Our intention is to use the XML specification file as input to a PN simulator, parsing the PNML language. By doing this, we intend to present another method to analyze and visualize the behavior pattern of each stage of the interaction process, mainly to solve problems in complex VEs. However, this graphical animation is only possible if there is a mechanism able to verify the correctness of the PN model. The use of this resource would allow performing PN validation before the automatic code generation and the PNML specification, providing consistency and completeness to the created model.

We are analyzing ways to verify the correctness during the model export process, since Dia is a general graphical editor and does not let the structural analysis take place in design time. This approach could allow for error detection within the editor, avoiding redraws and recoding during run-time and simulation-time stages. As a result, an accurate model could be generated, allowing the specific tools to verify the PN properties, through formal techniques to certify the absence of undesired system behaviors, such as deadlocks. Next, it would be necessary an evaluation session to validate these models, using one of the PN tools to analyze formally the system.

Another interesting idea would be to incorporate our methodology to a VR framework, presenting a complete development platform. Resources for analysis, project, development and evaluation of VE prototypes could be integrated in a single tool, allowing for the detection of faults in project time. With this in mind, frameworks such as VR Juggler [Vrjuggler, 10], MORGAN [Morgan, 10], and DIVERSE [Diverse, 10] are being analyzed, as they already use an extensive set of software modules that abstract devices, avatars and VEs. Our methodology could be adapted to function as one interaction framework integrated to existing resources, becoming an important feature of these tools.

Acknowledgements

This work was partially funded by Tecgraf, Computer Graphics Technology Group, at PUC-Rio. We are also grateful for the fellowships granted by Dell/PUCRS

Agreement and CAPES - the Brazilian Ministry of Education Agency. Alberto B. Raposo thanks FAPERJ for the individual support granted (#E-26/102.273/2009).

References

[Arroyo, 08] Arroyo, R. F., Gea, M., Garrido, J. L., Haya, P. A.: "Development of Ambient Intelligence Systems Based on Collaborative Task Models"; Journal of Universal Computer Science, 14, 9 (2008), 1545-1559.

[Basdogan, 00] Basdogan, C., Ho, C. H., Srinivasan, M. A., Slater, M.: "An Experimental Study on the Role of Touch in Shared Virtual Environments"; ACM Transactions on Computer-Human Interaction, 7, 4 (2000), 443-60.

[Billington, 03] Billington, J., Christensen, S., Van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: "The Petri Net Markup Language: concepts, technology, and tools"; In Proc. ICATPN'03, Lect. Notes in Comp. Sci. 2679, Berlin (2003), 483-506.

[Bowman, 99] Bowman, D. A., Hodges, L. F.: "Formalizing the Design, Evaluation, and Application of Interaction Techniques for Immersive Virtual Environments"; Journal of Visual Languages and Computing, 10, 1 (1999), 37-53.

[Bowman, 04] Bowman, D. A., Kruijff, E., LaViola, J. J., Poupyrev, I.: "3D User Interfaces: theory and practice"; Addison-Wesley, Redwood City (2004).

[Csisinko, 07] Csisinko, M., Kaufmann, H.: "Towards a Universal Implementation of 3D User Interaction Techniques"; In Proc. MRUI'07, Charlotte (2007), 17-24.

[Dia, 10] Dia: a drawing program, 2010, http://live.gnome.org/Dia.

[Diverse, 10] DIVERSE: Device Independent Virtual Environment – Reconfigurable, Scalable and Extensible: an open source virtual reality toolkit, 2010, http://diverse.sourceforge.net/diverse.

[Duval, 06] Duval, T., Lecuyer, A., Thomas, S.: "SkeweR: a 3D Interaction Technique for 2-User Collaborative Manipulation of Objects in Virtual Environments"; In Proc. 3DUI'06, Washington (2006), 69-72.

[Figueroa, 02] Figueroa, P., Green, M., Hoover, H. J.: "InTML: a Description Language for VR Applications"; In Proc. Web3D'02, Tempe (2002), 53-58.

[Jensen, 97] Jensen, K.: "Coloured Petri Nets: basic concepts, analysis methods and practical use"; Springer-Verlag, Berlin (1997).

[Lindeman, 99] Lindeman, R. W.: "Bimanual Interaction, Passive-Haptic Feedback, 3D Widget Representation, and Simulated Surface Constraints for Interaction in Immersive Virtual Environments"; School of Engineering and Applied Science, George Washington University, PhD. Thesis (1999).

[Margery, 99] Margery, D., Arnaldi, B., Plouzeau, N.: "A General Framework for Cooperative Manipulation in Virtual Enviroments"; In Proc. of the IEEE Virtual Environments, Los Alamitos (1999), 169-178.

[Martínez, 08] Martínez, D., García, A. S., Martínez, J., Molina, J. P., Gonzalez, P.: "A Model of Interaction for CVEs Based on the Model of Human Communication"; Journal of Universal Computer Science, 14, 19 (2008), 3071-3084.

[Memmel, 08] Memmel T., Reiterer, H.: "Model-Based and Prototyping-Driven User Interface Specification to Support Collaboration and Creativity"; Journal of Universal Computer Science, 14, 19 (2008), 3217-3235.

[Morgan, 10] MORGAN: a distributed multi-user framework for VR/AR applications, Fraunhofer FIT, 2010, http://www.fit.fraunhofer.de/MORGAN.

[Murata, 89] Murata, T.: "Petri Nets: properties, analysis and applications"; Proceedings of the IEEE, 77, 4 (1989), 541-580.

[Olwal, 04] Olwal A., Feiner, S.: "Unit: modular development of distributed interaction techniques for highly interactive user interfaces"; In Proc. GRAPHITE'04, Singapore (2004), 131-138.

[Palanque, 97] Palanque, P. A., Bastide, R.: "Synergistic Modeling of Tasks, Users and Systems Using Formal Specification Techniques"; Interacting with Computers, 9, 2 (1997), 129-153.

[Pinho, 08] Pinho, M. S., Freitas, C. M. S., Bowman, D. A.: "Cooperative Object Manipulation in Collaborative Virtual Environments"; Journal of the Brazilian Computer Society, 14, 2 (2008), 53-67.

[Riege, 06] Riege, K., Holtkamper, T., Wesche, G., Frohlich, B.: "The Bent Pick Ray: an extended pointing technique for multi-user interaction"; In Proc. 3DUI'06, Washington (2006), 62-65.

[Ruddle, 03] Ruddle, R. A., Savage, J. C., Jones, D. M.: "Levels of Control During a Collaborative Carrying Task"; Presence: Teleoperators and Virtual Environments, 12, 2 (2003), 140-155.

[Ruddle, 02] Ruddle, R. A., Savage, J. C., Jones, D. M.: "Symmetric and Asymmetric Action Integration During Cooperative Object Manipulation in Virtual Environments"; ACM Transactions on Computer-Human Interaction, 9, 4 (2002), 285-308.

[Sallnäs, 02] Sallnäs, E. L.: "Collaboration in Multimodal Virtual Worlds: Comparing Touch, Text, Voice and Video"; In The social life of avatars, Springer-Verlag New York (2002), 172-187.

[Smith, 99] Smith S., Duke, D.: "Using CSP to Specify Interaction in Virtual Environments", University of York, York, Technical Report YCS 321 (1999).

[Smith, 99] Smith S., Duke, D.: "Virtual Environments as Hybrid Systems"; In Proc. Eurographics UK 1999, Abington (1999), 113-128.

[Vitzthum, 06] Vitzthum, A.: "SSIML/Components: a Visual Language for the Abstract Specification of 3D Components"; In Proc. Web3D'06, Columbia (2006), 143-151.

[Vrjuggler, 10] The VR Juggler Suite, 2010, http://www.vrjuggler.org.

[Wieting, 96] Wieting, R.: "Hybrid High-Level Nets"; In Proc. WSC 1996, Coronado (1996), 848-855.

[Willans, 01] Willans, S., Harrison, M. D.: "Prototyping Pre-Implementation Designs of Virtual Environment Behaviour"; In Proc. EHCI'01, Lect. Notes in Comp. Sci. 2254, Berlin (2001), 91-108.

[Ying, 04] Ying, J., Gračanin, D.: "Petri Net Model for Subjective Views in Collaborative Virtual Environments"; In Proc. SG'2004, Lect. Notes in Comp. Sci. 3031, Berlin (2004), 128-134.