CNJ: A VISUAL PROGRAMMING ENVIRONMENT FOR CONSTRAINT NETS

Fengguang Song and Alan K. Mackworth
{fgsong, mack}@cs.ubc.ca
http://www.cs.ubc.ca/spider/{fgsong, mack}

Lab for Computational Intelligence
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4, Canada

ABSTRACT

The *Constraint Nets* (CN) model has proven to be useful for a wide variety of purposes, ranging from intelligent agent systems and real-time embedded systems, to integrated hybrid systems with various time structures [3]. In this paper, a new visual programming environment called CNJ (Constraint Nets in Java) using component-based technologies is described.

CNJ uses JavaBeans, Bean Introspection, drag-and-drop and Java Swing MDI (Multiple Document Interface) technologies, as well as XML as its standard interchange format. The tool supports CN modeling, simulation, and 3D animation for hybrid systems. Furthermore, it provides support for top-down design, middle-out design, and bottom-up design where the *module bean* can be reused anywhere in any other CN model, saving designers time and effort.

1. INTRODUCTION AND MOTIVATION

Constraint Nets (CN) was developed by Ying Zhang and Mackworth for modeling hybrid, dynamic systems [2][4][7]. CN is a generalization of dataflow with multiple data and events, in which a control system can be described as several block diagrams. A constraint net represents a set of equations, with *locations* as variables and *transductions* as functions [3]. Since it has inherent graphical tokens and hierarchies, *constraint nets* is an ideal model for visual programming. The CNJ system was motivated by the idea of realizing a visual environment to support *constraint nets* modeling.

Java is an object-oriented language and its Java Beans technology has the important characteristics of reusability, bean introspection and platform-independence. The introspecting tool of CNJ has the ability to discover what events a bean can fire and what properties a bean owns, and the ability to display and edit them dynamically. It uses Java's event handling mechanism (registration of event listeners) to connect *transductions* and *locations* and make them communicate with each other. The mechanism of the Java event successfully realizes support for various time structures of CN: discrete, continuous, or event-based. Java Swing's capabilities give the CNJ tool a useful GUI and the feature of MDI to support top-down and bottom-up modeling. Because CNJ is implemented in pure Java and component-based JavaBeans, it is not hard to change to a web-based application using EJB or RMI. We have also defined CNML (Constraint Nets Markup Language) as an XML-based interchange format for *Constraint Nets*. CNML uses Scalable Vector Graphics (SVG) 1.0 [12] to store 2D graphics, and Mathematical Markup Language (MathML) [13] to store a variety of functions of CN *transductions*. This characteristic makes it interchangeable with other XML tools easily.

Previous work on hybrid system modeling and simulation has focused on some specific time structures, but not on various ones. Simulink [1], based on Matlab, is a visual programming and simulation environment for both continuous and discrete dynamic systems. However, it doesn't support event-based time structure, which is very important for a hybrid dynamic system. Although it supports bottom-up modeling well (by grouping), it doesn't support top-down modeling which is very helpful for some users [1]. There has been some component-based simulation work done. [10][11][5] use Sun's BDK (Bean Development Kit) environment to model and simulate systems which actually imposes some limits on their design, for instance, the weak GUI which was implemented by Java AWT, the limited number of visual components (due to BDK's internal support) to represent bean properties, and the invisible event wires between beans. Also there are only Java-level events in them, with no concept of continuous time structure, which is essential for complex hybrid systems. Furthermore, to build a complex model based on customized Java Beans is very challenging. All of these limits make them difficult to be used practically. With CNJ, a specific environment for constraint nets, users are able to visually drag and drop CN nodes and connect them to model a hybrid system as well as to simulate it. In addition, with the XML-based CNML interchange format, it becomes fairly convenient to export to and import from other tools, and easy to read and understand, better than Java Bean's *serialization* mechanism [15].

2. CONSTRAINT NETS

Intelligent systems embedded as controllers in real systems or virtual systems are designed in an online model based on various time structures: continuous, discrete and event-based. However, the Constraint Satisfaction Problem (CSP) paradigm and Constraint Programming (CP) are inadequate for this kind of task, since they are primarily offline. The CN model is realized as an online dataflow-like distributed programming language with a formal algebraic denotational semantics, a specification language, and a real-time temporal logic [9].

In CN, an intelligent agent system is modeled as a symmetrically coupling of an agent with its environment as shown in figure 1.

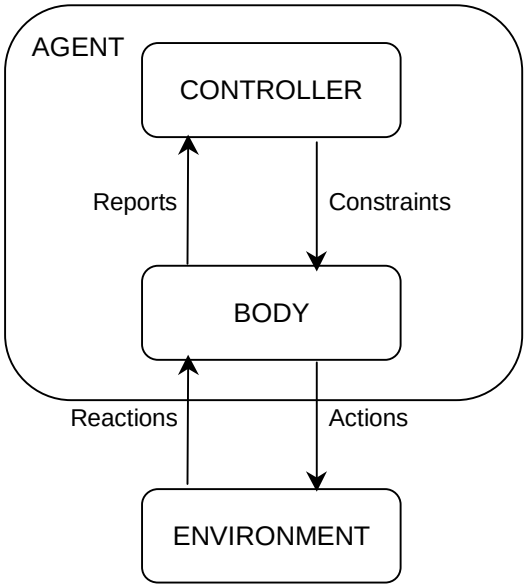

Figure 1. The structure of a constraint-based agent system

Both of the controller and the body consist of discrete-time, continuous-time or event-driven components operating over discrete or continuous domains. The controller has perceptual subsystems that can (partially) observe the state of the body, and through it, the state of the environment.

Intelligent systems are typically hybrid dynamic systems. *Constraint Nets* is used as an abstraction and a unitary framework to develop a hybrid dynamic system, analyze its behavior and understand its underlying physics. Formally, a constraint net is a triple *CN = <Lc, Td, Cn>*, where Lc is a finite set of locations, Td is a finite set of transductions, each with an output port and a set of input ports, Cn is a set of connections between locations and ports. Connections are restricted by the following: (1) there is at most one output port connected to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated.

A location can be regarded as a wire, a channel, a variable trace, or a memory cell. A transduction is a causal mapping from inputs to outputs over time, operating according to a certain reference time or activated by external events. Each module is a constraint net with a set of locations as its interface. A constraint net can be composed hierarchically using modular and aggregation operators on modules.

The semantics of the constraint net, with each location denoting a trace, is the least solution of the set of state evolution equations. A constraint net is depicted by a bipartite graph, where locations are depicted by circles, transductions by boxes, modules by round boxes, and connections by arcs.

3. OVERVIEW OF CNJ

Constraint Nets is a family of programming languages with a standard graphical representation [9]. To design a constraint net model, designers need to draw and define *transductions*, *locations*, *modules*, and *clocks* (a kind of transduction), and then wire them by *connections*.

CNJ is implemented in pure Java and can be run on different platforms. It has two main windows: `CNFrame` and `BeanPropertySheet` as shown in figure 2.

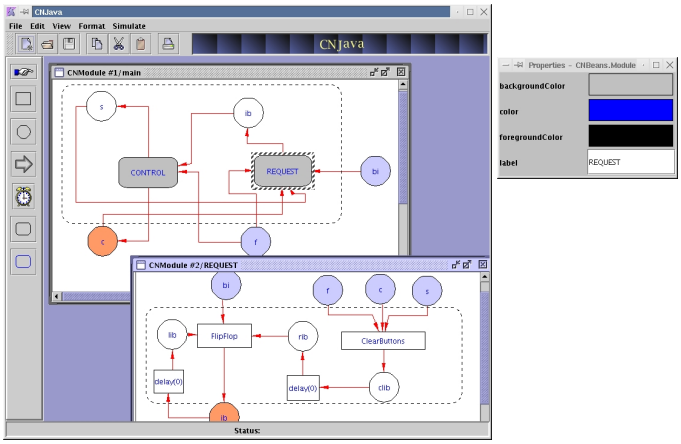

Figure 2. The GUI of CNJ

The `CNFrame` window is the bigger one. It consists of `MenuBar`, `ToolBar`, `ToolPane`, and `DrawPane`. `DrawPane` is the area for drawing constraint nets and has MDI to design different hierarchical levels of modules. `BeanPropertySheet` is the smaller one lying to the right close to `CNFrame`, which displays corresponding properties for a focused CN node in the `DrawPane` area. To design a constraint net model, you need to choose a graphical CN node from the `ToolPane`, then drag-and-drop it to the `DrawPane`. After that, you can customize the node's properties in the `BeanPropertySheet` window. You can use a *connection* to wire two CN nodes from the source's output port to the destination's input port.

Finally, you may want to simulate the model after compiling. The compiling step is able to discover some errors in the design of the CN model, such as uninitialized values, data type incompatibility, ill-defined transductions, and so forth. As for simulation, you can either watch the result from the model's output *locations*, or link it to a 3D animation package to see the 3D effect. We

implement the 3D animation package with the Java 3D API.

The CNJ tool supports building CN models in a hierarchical, modular and object-oriented manner. Users are able to design a model from bottom to top by grouping some components into a higher level module, or from top to bottom by first creating a module, and then designing the module in detail at a lower level. All the modules have the characteristic of reusability and can be used elsewhere by other CN models or modules. This component-based modeling and simulation environment makes building a CN model as simple as assembling and executing.

4. IMPLEMENTATION ISSUES

This section describes some actual implementation issues, as well as some results.

4.1 Constraint Net Nodes

A constraint net is composed of *transductions*, *locations*, *connections*, and *modules*. In CNJ, we call them constraint net nodes. All the CN nodes are implemented with JavaBean classes which are stored in their own *packages* (`CNBeans/Transduction, CNBeans/Location, CNBeans/Clock, and CNBeans/Module`). Since the nodes are sufficient for a constraint net, their corresponding bean classes are also enough to build a CN model.

A *module* is displayed as a visual Java bean (round rectangle) and has a popup child window to display its internal contents. Among the nodes, *transductions* are the most complicated. *Transductions* include *primitive transductions* (transliterations, delays) and *complex transductions* (event-driven transductions) [6][3]. To specify transductions' various functions, a class named `CNFunction` was implemented. We have provided some basic functions in CNJ, such as *plus*, *minus*, *times*, *and*, *or*, *polynomial function*, *piecewise*, *delay*, …. Users can create reusable complex modules based on the above basic *transductions* and reuse them in any other CN model.

4.2 GUI

As shown in Figure 2, the two main windows are arranged left and right. The left one is used to design and simulate constraint nets models, and the right one is used to display and edit the properties of the focused bean in the left window. Those GUI classes are stored in the `GUI` *package*, including `CNFrame, DrawPane, ToolPane, DrawFrame, BeanWrapper, PropertySheet,` and `PropertySheetPanel` classes. CNJ uses Sun's *JavaBeans Introspection* to discover a selected bean's properties and displays it dynamically in the `PropertySheet`. Using this method, we don't need to customize each dialog for every bean kind, instead, we only need to implement the *Introspection* mechanism for all the beans. Since we implemented the GUI in Java Swing, not in Java AWT, it is elaborate and more platform-independent .

4.3 Constraint Nets Markup Language (CNML)

CNML is an XML-based interchange format defined for Constraint Nets, which supports all kinds of constraint net models. In CNML, the model and the CN nodes are represented by XML *elements* [8][14]. The tags of the XML elements are named after *constraint net* concepts (e.g. `<Model>, <Module>, <Transduction>, <Location>, and <Connection>`). The following figures 3, 4 and 5 are examples of *transduction*, *location*, and *connection* represented by CNML.

```
<transduction id = "0">
   <graphics>
       <rect x = "   " y = "   " width = "   "
       height = "   " fill = "color" stroke =
       "color" />
   </graphics>
   <label color = "blue">
       +
   </label>
   <function>
       <input number = "2" />
       <math>
           <apply>
               <plus />
               <ci type = "integer"> input </ci>
               <ci type = "integer"> input </ci>
           </apply>
       </math>
   </function>
</transduction>
```

Figure 3. A CNML example for the "plus" transduction

```
<location id = "1">
  <graphics>
    <ellipse cx = "  " cy = "  " rx = "  " ry =
    "  " fill = "color" stroke = "color" />
  </graphics>
  <label color = "blue">
     input__x
  </label>
  <ci type = "integer" />
</location>
```

Figure 4. A CNML example for a location

```
<connection id = "2">
  <polyline points = " x1,y1 x2,y2 x3,y3 …" />
  <source> id </source>
  <target> id </target>
</connection>
```

Figure 5. A CNML example for a connection

We use the SVG1.0 [12] specification to represent constraint nets' graphical information, and MathML2.0 [13] to represent transductions' functions. This makes CNML compatible with W3C's standard. We use JDOM beta 7 [16] to handle the CNML files, to load or save, and to export to or import from other file format, i.e., HTML, XHTML and Simulink's MDL. The CNMLIO class is implemented to support handling CNML files.

4.4 Simulation

CNJ simulation is based on the Java Event mechanism. Event is a core feature of Java, which allows some components to act as event sources for event notifications that can be caught and processed by some other listeners. In CNJ, when users draw a *connection* from one output port of a node to another input port of another node, the former node works as an *event source* and the latter works as an *event listener*. Every *transduction* has a unique *clock transduction* input. Clocks fire clock tick events to their connected *transductions*. Whenever a transduction receives a clock tick event, it computes the result from its input traces right away, then outputs a trace to its connected *location*. A constraint net model may have several *clocks* with different frequencies (Hz). It might have one *global clock* and several *local clocks* distributed in different areas or modules. By connecting to a *clock* with an infinitely high frequency, *continuous time structure* can be realized virtually.

5. A CAR MODEL IN CONSTRAINT NETS

In our Laboratory for Computational Intelligence, a testbed has been installed for radio-controlled cars playing soccer [4]. Each "soccer player" has a car-like mobile base. It can move forward and backward with a throttle setting, and can make turns by steering its two front wheels.

Figure 6 illustrates the configuration of a car. Let v be the velocity of the car and α be the current steering angle of the front wheels. v and α, for now, can be considered as control inputs to the car. The dynamics of the car can be simply modeled by the following differential equation:

$$\dot{x} = v\cos(\theta),\ \dot{y} = v\sin(\theta),\ \dot{\theta} = v / R \qquad (1)$$

where (x, y) is the position of the tail of the car, θ is the heading direction and $R = L / \tan(\alpha)$ is the turning radius given the length of the car L. The controller of such a car is equipped with both digital and analog devices.

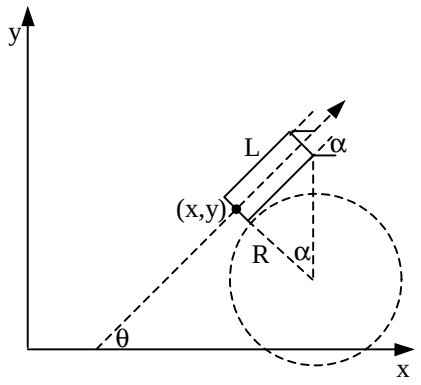

Figure 6. The configuration of a car

The dynamics of the car is modeled as a constraint net with CNJ, as shown in Figure 7, in which cosine, sine, tan, +, *, and / are transliterations, v and alpha are input locations, x, y, and theta are output locations.

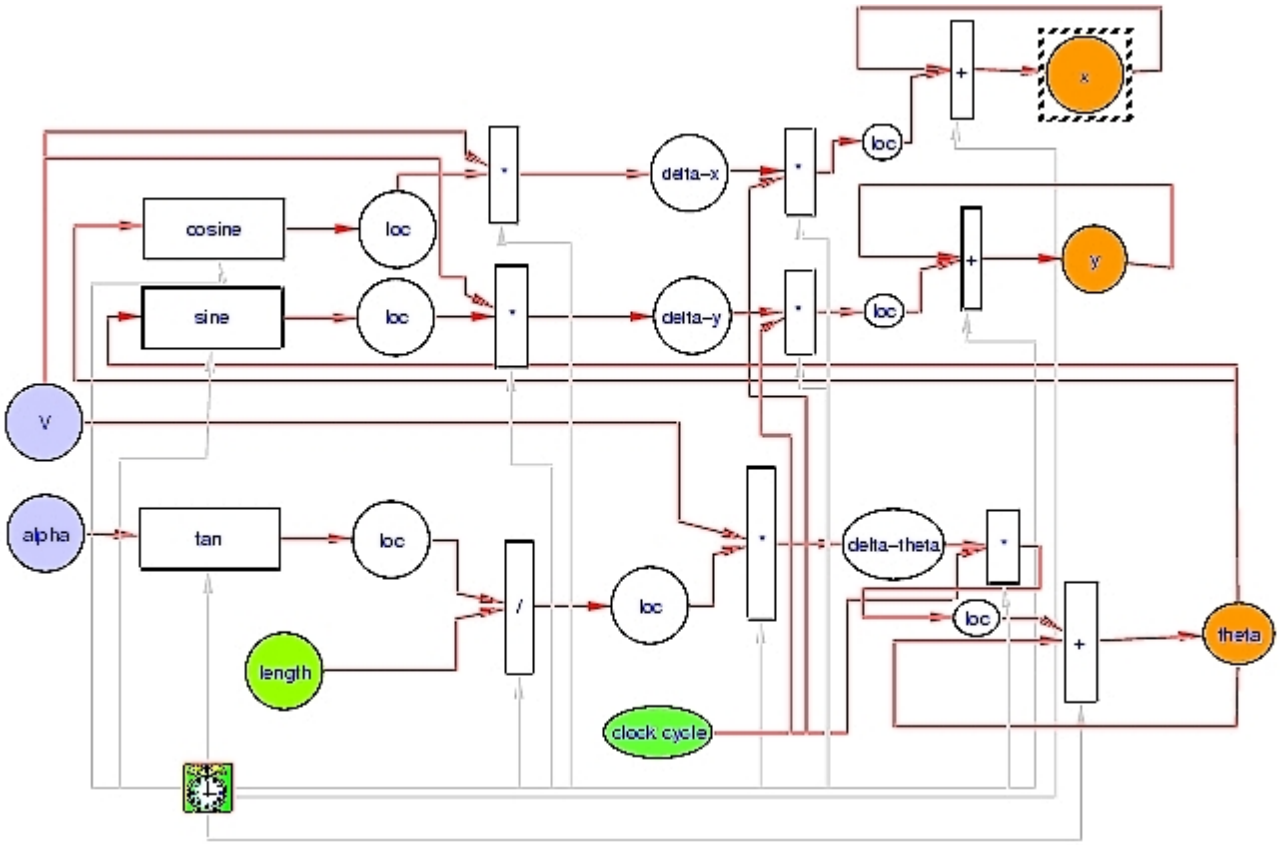

Figure 7. The constraint net of equation (1)

The clock-shaped rectangle is a global clock transduction. It fires a clock event to every connected transduction after a period of time equal to its cycle time. The location named "clock cycle" has the same value as the clock's cycle time period. Essentially it is a discrete/event-based model, but if we set the clock's frequency to twice the user's required frequency, we can consider it continuous.

6. RESULTS

To understand CNJ's real-time response, we ran CNJ to simulate some CN models under the environment of Linux Redhat7.1 with PIII 1G Hz, 256M SDRAM, J2SDK1.4. Although CNJ clocks are not perfectly accurate, we find them quite good. For CNJ clocks running at up to 100Hz, the model can run at close to real-time. Moreover, from the user's point of view, the response lag time is acceptably low, even when implemented in pure Java (Swing).

Our work proves that JavaBeans and Java Event mechanisms are efficient for real-time simulation and realization of reusability. The visual programming environment is helpful for users.

7. CONCLUSION

This paper presents our work in modeling and simulating real-time hybrid dynamic system with *constraint nets*. It uses the technologies of component-based software and Java Beans to successfully implement a practical visual real-time visual environment: Constraint Nets in Java (CNJ). The reusability of beans makes building constraint net models easier, and allows users to reduce time and effort in building similar models. In addition, the XML-based interchange format CNML makes translation to and from other tools feasible and convenient. Constraint Nets also includes a specification language (two versions: Timed Linear Temporal Logic and Timed ∀-automata), that allows the designer to specify and verify control systems. Our development plans, therefore, include an extension to support graphical specification and verification tools.

ACKNOWLEDGEMENTS

We thank Yu Zhang, Ying Zhang, Valerie McRae, and Lan Lin for valuable help. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

REFERENCES

1. B. Shahian, M. Hassul: 1993, "Control System Design Using Matlab", Prentice-Hall. Inc..
2. Zhang, Y. and A. K. Mackworth: 1994, "Specification and Verification of Constraint-Based Dynamic Systems". In: A. Borning (ed.): Principles and Practice of Constraint Programming, No. 874 in Lecture Notes in Computer Science. Springer-Verlag, pp. 229 - 242.
3. Zhang, Y.: 1994, "A Foundation for the Design and Analysis of Robotic Systems and Behaviors". Ph.D. thesis, University of British Columbia, Vancouver, British Columbia.
4. Zhang, Y. and A. K. Mackworth: 1995, "Synthesis of Hybrid Constraint-Based Controllers". In: P. Antaklis, W. Kohn, A.Nerode, and S.Sastry (eds.): Hybrid Systems II, Lecture Notes in Computer Science 999. Springer Verlag, pp. 552 - 567.
5. John A. Miller, Y. Ge, J. Tao: 1998, "Component-Based Simulation Environments: JSIM As a Case Study Using Java Beans". In: Proc. 1998 Winter Simulation Conference pp, 373 - 381.
6. Zhang, Y. and A. K. Mackworth: 1999, "Modeling and Analysis of Hybrid Systems: An Elevator Study". In: H. Levesque and F. Pirri (eds.): Logical Foundations for Cognitive Agents. Berlin: Springer, pp. 370 - 396.
7. A. K. Mackworth: 2000, "Constraint-Based Agents: The ABC's of CBA's". In: Proc. 6th Int. Conf. On Principles and Practice of Constraint Programming - CP2000. Springer LNCS 1894, Singapore, pp. 1 - 10.
8. M. Jungel, E. Kindler, M. Weber: 2000, "The Petri Net Markup Language". In: S. Philippi(Ed.): 7. Workshop Algorithm for Petri Net.
9. A. K. Mackworth and Zhang, Y.: 2001, "Constraint-Based Agents: A Formal Model for Agent Design". In: UBC Computer Science Technical Report TR-2001-09.
10. Y. Wang and S. Ho: 2001, "Implementation of a DEVS-JavaBean Simulation Environment". In: 2001 Advanced Simulation Technologies Conference.
11. K. Verschaeve, B. Wydaeghe, F. Westerhuis: 2001, "Visual Composition with SDL Beans". In: Proceedings of ECBS 2001, Washington, USA.
12. W3C: September 2001, "Scalable Vector Graphics (SVG) 1.0 Specification".
13. W3C: 2001, "Mathematical Markup Language (MathML) Version 2.0".
14. E. R. Harold: 2001, "XML Bible 2nd edition", IDB Books.
15. Andy Quinn: 2001, "Trail: JavaBeans(TM)", http://java.sun.com/docs/books/tutorial/javabeans/index.html
16. JDOM Beta 7: 2001, http://www.jdom.org/

AUTHOR BIOGRAPHIES

Fengguang Song is currently a graduate student in Department of Computer Science at the University of British Columbia. He received the M.Sc. in Computer Science from Nanjing University of Aeronautics and Astronautics, P.R. China in 1999. His research interest includes constraint-based real-time systems, software engineering, and image-based modeling and rendering. His e-mail address is <fgsong@cs.ubc.ca> and URL is <http://www.cs.ubc.ca/~fgsong>.

Alan K. Mackworth is a Professor of Computer Science at the University of British Columbia and holds a Canada Research Chair in Artificial Intelligence. His research focuses on the theory and applications of constraint-based dynamic systems, computational vision and robotics, and hybrid systems. His e-mail address is <mack@cs.ubc.ca> and URL is <http://www.cs.ubc.ca/~mack>.