Towards Support for Verification of Adaptative Systems with Djnn

Daniel Prun, Mathieu Magnaudet, Stéphane Chatty

Towards Support for Verification of Adaptative Systems with Djnn

Daniel Prun, Mathieu Magnaudet, Stéphane Chatty
Université de Toulouse - ENAC
Toulouse, France
e-mail: {daniel.prun, mathieu.magnaudet, chatty}@enac.fr

***Abstract*—Djnn is a general framework dedicated to the development of complex interactive systems. We describe ongoing work aimed at developing verification mechanisms through the definition of syntax, grammar and semantics for djnn models. The results will serve to perform formal verification of interactive systems.**

Keywords-interactive system; component; control structure; model; syntax; semantic; formal verification.

I. INTRODUCTION

For more then 30 years, dedicated languages and methods have been designed and used to deal with the development of critical systems (transportation, health, nuclear and military systems). These languages and methods are used for the development of safe, functionally correct systems. For example, VHDL (VHSIC Hardware Description Language) [1] is hugely used for the development of hardware circuits, SCADE (Safety Critical Application Development Environment) [2] language is used for control and command systems.

However, highly interactive and adaptive systems have recently and progressively appeared [3], [4]. For example, air traffic control systems, surveillance systems or automotive systems have to react to many event sources: user events (from classic keyboard/mouse to more advanced interaction means such as multi-touch surfaces, gesture recognition and eye gaze), pervasive sensors, input from other subsystems, etc.

Difficulties have been observed in using existing languages and methods on these kinds of systems. Indeed, these systems require new control structures in order to manage dynamicity or to support different design styles, such as state machines and data flows, and when using existing languages this often leads to problems in the software architecture [5], [6]. We argue that part of these issues are due to the lack of a well-defined language for representing and describing interactive software design in a way that allows, on the one hand, system designers to iterate on their designs before injecting them in a development process and on the other hand, system developers to check their software against the chosen design.

This paper describes a work in progress within the development of a general framework (named Djnn) dedicated to the development of interactive systems. Section II presents the current state of Djnn and introduces requirements for its supporting systems verification. Section III discusses the early results obtained so far in the context of HoliDes (Holistic Human Factors and System Design of Adaptive Cooperative Human Machine System) project. Section IV concludes with description of future developments.

II. DJNN

Djnn [7] is a general framework aimed at describing and executing interactive systems. It is an event driven component system with:

- a unified set of underlying theoretical concepts focused on interaction,
- new architectural patterns for defining and assembling interactive components,
- support for combining interaction modalities,
- support for user centric design processes (concurrent engineering, iterative prototyping).

A. Control primitives

Djnn relies on a fundamental control primitive called "binding". A binding is a component that creates a coupling between two existing components. If there is a binding between components C1 and C2, then whenever C1 is activated, C2 is activated (C1 is called trigger and C2 is called action). A binding can be interpreted as a transfer of control, like a function call in functional programming or a callback in user interface programming. Figure 1. shows examples of binding definitions.

```
# beeping at each clock tick
binding (myclock, beep)

# controlling an animation with a mouse button
binding (mouse/left/press, animation/start)
binding (mouse/left/release, animation/stop)

# quitting the application upon a button press
binding (quitbutton/trigger, application/quit)
```

Figure 1. Examples of bindings definitions in Djnn.

Bindings can be used to derive a set of control structures required to describe interactive softwares: Finite State Machine (FSM), Connector (used to transfer data between two components), Watcher (allow to connect C1 and C2 to C3 where C3 is activated only when C1 and C2 are synchronously activated) or Switch (activates one of several components according to input data values). Figure 2. shows examples of derived control structure definitions.

 ISBN: 978-1-61208-390-2

```
# ensure that rectangle rect1 will move with
# the mouse.
connector (mouse/position/x, rect1/position/x)
connector (mouse/position/y, rect1/position/y)

# m is performed when input1 and anput2 are
# simultaneously activated
multiplication m (input1, input2, output)
watcher (input1, input2, m)
```

Figure 2. Examples of derived control structure definitions in Djnn.

FSMs are one of the most used control structures for describing user interfaces with Djnn. They contain other components named states and transitions. Transitions are bindings between two states (named origin and destination). A transition is active only when its origin is active. It behaves as a binding with a default action: changing the current state of the FSM to its destination state. Therefore, the transitions define the inputs of the state machine: the state evolves on the sequence of activation of the triggers of the transitions, and ignores events that do not match the current state. Figure 3. shows the internal behavior of a software button designed for use with a mouse: the Djnn code above implement the FSM shown at the bottom. r is the graphical representation of the button (a rectangle component).

```
component mybutton {
 rectangle r (0, 0, 100, 50)
 fsm f {
   state idle, pressed, out
   transition press(idle, r/press, pressed)
   transition trigger(pressed, r/release, idle)
   transition leave (pressed, r/leave, out)
   transition enter (out, r/enter, pressed)
 }
}
```

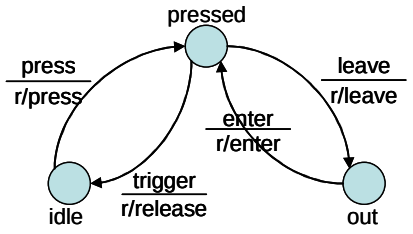

Figure 3. Example of a FSM definition.

B. An architecture of reactive components

In Djnn, every entity you can think of, abstract or physical, is a component. In addition to the control structures introduced above, Djnn comes with a collection of basic types of components dedicated to user interfaces: graphical elements, input elements (mouse, multi-touch, sensors, etc.), file elements, etc. Every type of component can be dynamically created or deleted.

To design interactive systems, components must be interconnected and organized. Interconnection is obtained with control structures, and can be performed independently of the nature and location of components. For example, a binding can connect the position of a mouse press to the position of a rectangle, so that the rectangle moves whenever the mouse is pressed. Structuration is obtained with a dedicated control structure: the parent-child interconnection that allows creating a hierarchy of components. For example, a complex graphical scene is composed of several graphical sub-components; a mouse is made of two buttons and one wheel; a FSM is made of several bindings etc. The designer can explicitly manage this tree-oriented architecture.

Combining the tree structure and the other control structures can be used for creating complex interactive behaviors and not only graphical scenes. For instance, combining FSMs by coupling their transitions, or by controlling the activation of one by a state or a transition of another, makes it possible to create complex behaviors (see example in Figure 4.). The tree structure also makes it easier to structure applications as collections of reusable components.

```
# connect “trigger” transition to a component
action “quit”
binding (mybutton/trigger, application/quit)
```

Figure 4. Example of connection with FSM.

Whenever a composite component is activated, the activation of its children components is iteratively performed. Each visited component is then activated and eventual transversal connections are activated.

C. Djnn in use: realizations and limitations

Djnn components can be created with various programming languages (Perl, Python, C, C++ or Java) or loaded from XML (Extensible Markup Languages) files. For instance, complex graphic scenes can be loaded from SVG (Scalable Vector Graphics) files. The final application is then compiled and linked with specific Djnn libraries. Dedicated available target are Windows, Linux or Mac OSX platform.

Djnn has been used for several realizations related to complex interactive systems. For example, in [8], Djnn has been used for the design and implementation of a ground control station for squads of civil Unmanned Aerial Vehicles (UAVs) (see Figure 5.). With Djnn, programming user interface adaptation comes down as a special case of programming interactive behavior. This allowed to easily implement many scenarios of adaptation, from simple state transition to complex graphical reconfigurations triggered by heterogeneous event sources. Thus, we have been able to demonstrate that Djnn provides a suitable framework to develop complex adaptive interfaces.

In [10], Djnn has been used to develop a prototype of a drawing tool overlapped on top of maps in a maritime surveillance system. The tool is used to share information between the crew during search and rescue missions. This example demonstrated how Djnn facilitates the development of user interfaces by offering a support for rapid prototyping and iterative processes.

 ISBN: 978-1-61208-390-2

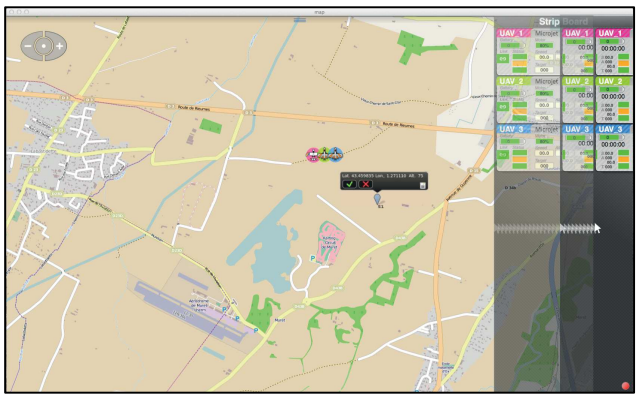

Figure 5. Djnn used for the design of a UAVs squad control.

Djnn is the visible result of an ongoing research project on programming languages for interactive systems. So far, verification of interactive systems designed with Djnn has not been studied. Focus has been put on the development of the implementation of the framework. Clearly, Djnn lacks several elements to enable the development of critical interactive systems:

- #1: a formal syntax and semantic for Djnn models,
- #2: mechanisms to translate Djnn applications into languages supporting model checking simulation or formal verification such as Event B [12] or Spin/Promela [11],
- #3: mechanisms to perform property verification directly on Djnn models.

Note that #1 is a prerequisite: without formal semantic, there is no possibility for verification. In the next section, we present our first results in this direction.

III. DJNN IN HOLIDES

The research results presented in this section are part of the HoliDes project, whose main goal is to design adaptive cooperative systems, focusing on the optimization of the distribution of workloads between humans and machines [9]. During the first year of this project, Djnn has been improved to prepare it for verification of interactive systems along two axes:

- Specification of the syntax and grammar through XML formats,
- Development of a formal semantic in Petri Nets.

A. *Syntax and grammar*

An abstract syntax and a grammar for Djnn have been defined through an XML schema. The model addresses most components available in Djnn, particularly control primitives. For example, Figure 6. contains the description of a binding and a FSM: a binding is an extension of a component containing identification of a source (“trigger”) and of a target (“action”). A FSM is an extension of a component containing a sequence of minimum of two states and a sequence of a minimum of one transition (state and transition are defined elsewhere in the XML schema).

```
<xs:complexType name="binding">
  <xs:complexContent>
    <xs:extension base="cmn:core-component">
      <xs:attribute name="source"
          type="xs:string" use="required" />
      <xs:attribute name="action"
          type="xs:string" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="fsm">
  <xs:complexContent>
    <xs:extension base="cmn:core-component">
      <xs:sequence>
        <xs:element name="state"
                    type="state"
                    minOccurs="2"
                    maxOccurs="unbounded" />
        <xs:element name="transition"
                    type="transition"
                    minOccurs="1"
                    maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 6. Djnn binding and FSM control structures described by the XML schema.

The main advantages provided by these definitions are:

- Definition of a well-defined model for Djnn: illicit constructs using the language can easily and automatically be detected during edition of the model thanks to the XML schema.
- Improvement of interoperability: this evolution is a first step towards the definition of a better integrated tool chain with the capability to dump a concrete GUI (Graphical User Interface) in an XML file, and conversely, to load and to execute a GUI from an XML based description.

B. *Towards a formal semantic*

Semantic of Djnn model is expressed through colored Petri Nets [13] extended with reset arcs [15]. We chose this formalism because, at a first glance, it offers good characteristics to represent both static and dynamic concerns through a state-transition semantic. It also allows to model simple data. All Djnn components are currently being individually modeled with Petri Nets. Figure 7. and Figure 8. give overviews of the semantic. The left part of Figure 7. represents a binding between a source and an action with a simple and unique transition. As a binding is a component, its interface also offers run and stop operations. The right part represents a connector: when activated, input data <X> is copied to the output. Figure 8. shows Petri Nets model of the button as defined in Figure 3.

 ISBN: 978-1-61208-390-2

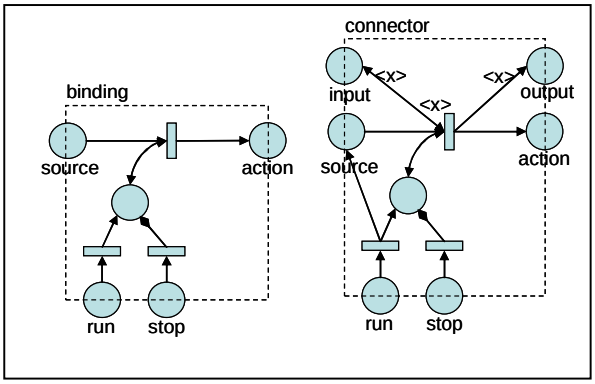

Figure 7. Models of a binding and a connector.

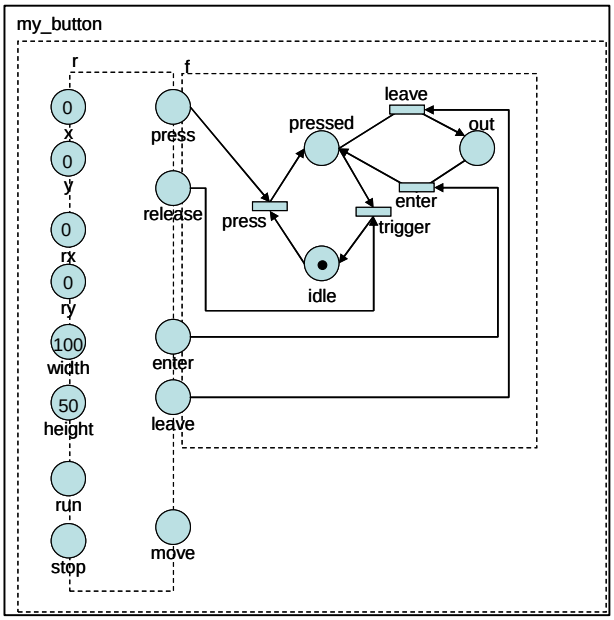

Figure 8. Model of FSM (as defined in Figure 3.

Composition of components is achieved through the merging of places of Petri Nets. This model of composition, even if it is asynchronous, seems to perform best for our purpose.

Such a formal definition of the semantic is central for verification purpose because:

- semantic of Djnn is no longer subject to misunderstandings or interpretations. A Djnn model has the same meaning for every actor in interaction with it (designer, code developer, final user, etc.);
- as the Petri Nets semantic is formal, several mathematical verifications become enabled: for example, LTL (Linear Temporal Logic) or CTL (Computation Tree Logic) properties [14], liveness or boundness properties. Moreover, translations to other languages specialized on formal verification become possible.

IV. CONCLUSION AND FUTURE PLANS

In this paper, current research related to a framework for the development and the verification of interactive safety critical systems has been presented. Although bases have already been developed (syntax and grammar through a XML schema, part of the semantic with Petri Nets), investigations remain to be done:

- So far, Petri Nets have showed their capability to model Djnn elements and mechanisms but some further analysis must be done on dynamic aspects of Djnn (creation/destruction of components).
- Use of the Petri Nets models to perform verification through simulation or through model analysis.
- Connections with tools specialized in formal verification.

Application on some real use cases, hopefully brought by HoliDes project, are also planned for the next phases.

V. ACKNOWLEDGMENTS

This research has been performed with support from the EU ARTEMIS JU project HoliDes (http://www.holides.eu/) SP-8, GA No.: 332933. Any contents herein reflect only the authors' views. The ARTEMIS JU is not liable for any use that may be made of the information contained herein.

REFERENCES

[1] "VHDL Language Reference Manual", IEEE Std 1076-2008.

[2] Scade homepage, http://www.esterel-technologies.com, [retrieved: 02, 2015]

[3] L. Bass et al. "The Arch model: Seeheim revisited", CHI'91 User Interface Developers Workshop, Apr. 1991.

[4] G.E. Pfaff, "User Interface Management Systems," Eurographics Seminars, Springer-Verlag, 1985.

[5] B. A. Myers, "Separating application code from toolkits: Eliminating the spaghetti of callbacks," In Proc. UIST, 1991, pp. 211–220, Addison-Wesley.

[6] B. A. Myers and M. B. Rosson, "Survey on user interface programming," In Proc. CHI, 1992, pp. 195–202, ACM Press.

[7] Djnn project homepage, http://djnn.net, [retrieved: 02, 2015].

[8] M. Magnaudet and S. Chatty, "What should adaptivity mean to interactive software programmers?" EICS 2014, ACM SIGCHI, Rome, Italy, Jun 2014, pp 13-22.

[9] HoliDes (Holistic Human Factors and System Design of Adaptive Cooperative Human Machine System) R&D project www.holides.eu, [retrieved: 02, 2015].

[10] C. Letondal, P. Pillain, E. Verdurand, D. Prun, and O. Grisvard, "Of Models, Rationales and Prototypes: Studying Designer Needs in an Airborne Maritime Surveillance Drawing Tool to Support Audio Communication," In Proc. of BCS HCI, ACM, 2014, pp. 92-102.

[11] G.J. Holzmann, "The Spin Model Checker: Primer and Reference Manual," 2003, Addison-Wesley.

[12] J.-R. Abrial, "Modeling in Event-B: System and Software Engineering," May 2010, ISBN: 9780521895569.

[13] K. Jensen, "Coloured Petri Nets," Berlin, Heidelberg, 1996, ISBN 3-540-60943-1.

[14] C. Baier and J.-P. Katoen, "Principles of Model Checking," 2008, The MIT Press.

[15] C. Dufourd, A. Finkel, and P. Schnoebelen, "Reset nets between decidability and undecidability," In 25th ICALP, vol. 1443 of LNCS, Springer, July 1998, pp. 103-115.

 ISBN: 978-1-61208-390-2