# Integrated Coding and Design in Software Production

C. Gonçalves, J. Soares and G. Barroso

***Abstract*—Colored Petri Nets (CPNs) can be used as a cornerstone evolutionary approach for software prototyping. In this work, CPNs are not only a resource for design, but also for developing high-level software. The coding process is completely integrated with the modeling, analyzing and tests phases. The use of prototypes, generated automatically with the CPN design, can avoid error propagation to the subsequent steps of software development, increasing quality and improving productivity. The main contributions of this work are: (i) the *game*-Aided by Modelling (GAM) method, which not only serves to model, but to create simple *games* from CPNs; and (ii) the Classes-Aided by Modelling (CAM) method generalizes GAM method to the object-oriented classes implementation. Two applications were developed to demonstrate the effectiveness of this approach and are also discussed here.**

***Index Terms*—Casual *games*, Colored Petri nets, Software models and Software prototyping**

## I. Introdução

TÉCNICAS para lidar com a complexidade de problemas computacionais baseiam-se usualmente no uso de abstrações. Uma estratégia recorrente para o desenvolvimento de software é fazer uso de diagramas que, além de auxiliar na representação do problema, podem especificar múltiplas perspectivas para o espaço da solução. Adicionalmente, diagramas são usados para análise prévia da solução e podem servir de instrumentos para simulação e para testes, antes da implementação.

Apesar de parecer óbvia a vantagem de utilização dos diagramas, alguns métodos de construção de software, tais como os Ágeis, questionam o valor de grandes volumes de artefatos intermediários. Isso inclui os diagramas da *Unified Modeling Language* (UML) [1] ou outro tipo de formalização, principalmente, quando usados como maneira de especificação antecipada. A justificativa é que, em geral, a natureza dos produtos de software só é amadurecida, ao longo do processo de desenvolvimento, podendo implicar em muito retrabalho sobre os referidos artefatos. Além disso, um dos princípios fundamentais do manifesto ágil destaca que "*software funcional é a medida primária de progresso no desenvolvimento*" [2]. Nessa perspectiva e considerando os requisitos de altíssima produtividade do mercado de software, muitas equipes ágeis utilizam pouca ou nenhuma modelagem, ou a utilizam de maneira muito restrita para discussões e elaborações rápidas de soluções [3].

Como forma alternativa à modelagem para lidar com a complexidade e trabalhar melhor a compreensão e a evolução de requisitos, é frequente o uso da prototipação. Essa técnica pode ser utilizada em duas formas, sendo a primeira conhecida como *throwaway* [4], em que o protótipo é unicamente usado para estudar aspectos do sistema, podendo ser completamente descartado em fases subsequentes do desenvolvimento. Embora a abordagem *throwaway* permita melhor compreender os requisitos, considerando que muito trabalho de programação pode ser empregado para a construção do protótipo, pode-se colocar em perspectiva o reaproveitamento do código gerado em etapas subsequentes. Dessa maneira, a segunda abordagem de prototipação denominada *cornerstone* [4], preconiza a evolução do protótipo sob um rigoroso sistema de correção de falhas para se tornar parte do produto final, permitindo aliar de maneira efetiva as fases de design e de construção do software. Entretanto, mesmo a utilização de protótipos não prescinde necessariamente da utilização de modelagem. Como a natureza dos produtos de software é variável e não pode ser generalizada, entende-se que não se deve negligenciar o valor de modelos como coadjuvantes no processo de construção de software.

Neste sentido, o objetivo principal deste trabalho é apresentar um método inspirado na prototipação evolucionária *cornerstone* que utiliza Redes de Petri Coloridas (RPCs) não apenas como um recurso para design, mas também para construção de software de alto nível, sendo código e modelos de RPCs completamente integrados na construção do software alvo. É discutido mais profundamente como as RPCs podem facilitar o desenvolvimento de software, utilizando o domínio dos *games* casuais como um estudo de caso. Entretanto, o uso de RPCs para especificação, validação e construção de *software* de alto nível de uma maneira integrada pode ser generalizado para softwares de outras naturezas. É também apresentada uma proposta de como uma classe do paradigma de orientação a objetos pode ser gerada via RPCs, generalizando o uso de RPCs na especificação, na validação e na construção de software de alto nível. Tal técnica atribui aos modelos as bases do desenvolvimento do software e ainda permite que o software (ou parte dele) seja executado de maneira imediata, permitindo a sua validação e, assim, atuando como medida primária de progresso no desenvolvimento. Na Seção II, é apresentada uma breve descrição sobre as definições e uso das RPCs. Os trabalhos relacionados são discutidos na Seção III. Os métodos de desenvolvimento de *games* e classes baseados em RPCs são propostos e discutidos na Seção IV. O item VI

Carlos H. R. Gonçalves, IFCE Departamento de Telemática, Av. Treze de Maio, 2081 - Benfica, Fortaleza - CE, 60040-531, Brasil (e-mail: hairon@ifce.edu.br).

José M. Soares, UFC PPGETI, Campus do Pici, Bloco 725, s/n-Pici, Fortaleza- CE, 60455-970 (e-mail: marques@ufc.br).

Giovanni C. Barroso, UFC PPGETI, Campus do Pici, Bloco 725, s/n - Pici, Fortaleza - CE, 60455-970 (e-mail: gcb@fisica.ufc.br).

trata das conclusões e dos trabalhos futuros. Por fim, as referências são expostas.

## II. Breve Descrição sobre Redes de Petri Coloridas

As RPCs definem uma linguagem gráfica para a construção de modelos de sistemas a eventos discretos e análise de suas propriedades. RPCs podem ser definidas como uma linguagem de modelagem que combina as capacidades das Redes de Petri (RP) com os recursos de uma linguagem de programação de alto nível [5]. A linguagem de programação das RPCs usada neste artigo é a *Colored Petri Net ML* (CPN ML), que se baseia na linguagem de programação funcional *Standard ML* [6]. As RPCs permitem a construção de modelos compactos e parametrizáveis. A vantagem das RPCs sobre as Redes de Petri (RP) tradicionais, conhecidas como redes de Petri lugar-transição, é a capacidade de modelar sistemas complexos e fornecer modelos com um alto nível de abstração e visualização [5]. As RPCs também contemplam análise temporal. Isto significa que as RPCs podem ser aplicadas para a análise de desempenho baseada em simulação para investigar métricas, tais como atrasos, taxa de transferência, tamanhos de fila, modelagem e validação de sistemas de tempo real. O conceito de módulos em RPC baseia-se em um mecanismo de estruturação hierárquica, o que permite que um módulo tenha submódulos que podem ser reutilizados em diferentes partes do modelo. Na Fig. 1, é mostrada a estrutura principal de um modelo RPC, constituído por lugares, transições e arcos. Com RPC, é possível usar tipos de dados e manipular dados complexos.

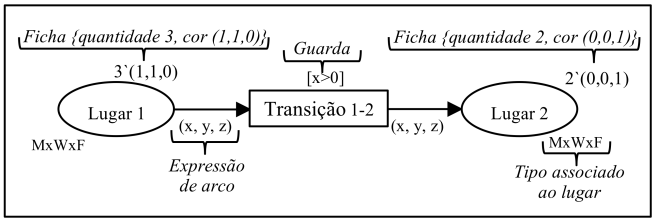



Fig. 1. Representação de uma rede de Petri colorida.

Em um modelo RPC, os lugares são representados por elipses. Cada lugar pode ser marcado com uma ou mais fichas e cada ficha tem um valor de dados ligado a ela. Esse valor de dados é chamado cor da ficha. É o número de fichas e as cores das fichas nos lugares individuais que, em conjunto, representam o estado do sistema modelado, o que é chamado de uma marcação do modelo RPC. As fichas em um lugar específico constituem a marcação daquele lugar. As transições representam os eventos que podem ocorrer no sistema e são representadas por retângulos. Quando a transição ocorre, ela remove fichas de seus lugares de entrada (aqueles lugares que têm um arco que conduz à transição) e adiciona fichas aos seus lugares de saída (os lugares que têm um arco proveniente da transição). As fichas que são removidas dos lugares de entrada e adicionadas aos lugares de saída, quando ocorre uma transição, são determinadas por meio das expressões de arco, que são as inscrições textuais posicionadas ao lado dos arcos individuais. As expressões de arco são escritas na linguagem de programação funcional CPN ML e são construídas a partir de variáveis, constantes, operadores e funções. Quando todas as variáveis em uma expressão são ligadas a valores do tipo correto, a expressão pode ser avaliada. As transições possuem uma guarda, que é uma expressão booleana. Quando uma guarda está presente, ela deve ser avaliada como verdadeira para que a transição esteja habilitada e possa ocorrer.

## III. Trabalhos Relacionados

Há na literatura vários trabalhos que utilizam RP para modelagem de *games*, mas trabalhos que utilizam RPCs são mais escassos. Não foram encontrados trabalhos que proveem mecanismos para prototipação visual e interativa de software com resultados consistentes. Lee e Cho propõem em [7] um método baseado em RP para especificar pequenas missões de personagens de um jogo de interpretação de papéis (*Role-Playing game* - RPG). Nesse caso, as RPs são usadas para mapear os eventos que consistem nas ações e nas interações do jogador com o ambiente. A evolução desse trabalho apresenta uma ferramenta chamada de PlotWizard [8] que serve para criar a trama de um *game*. Entretanto, o formalismo, apresentado por esses autores, suporta apenas RPs para especificar como os personagens interagem, não interferindo nas propriedades gráficas e de controle do *game*. Essa proposta não tem um caráter geral na área de *game*, estando limitada aos RPGs. Em [9] são apresentadas RPs para modelar um jogo em que um navegador português vai em busca de conquistar novas terras, ora tendo de cooperar com os nativos, ora tendo que lutar contra eles. São discutidas vantagens e desvantagens de modelar jogos utilizando-se de RPs. Dentre as principais vantagens estão a simplicidade de usar um único diagrama para modelar os diversos personagens e os comportamentos do jogo, bem como a facilidade de análises e testes, mesmo antes da implementação. Dentre as desvantagens está a complexidade das RPs, quando há um número muito grande de lugares, de transições e de arcos. Essas desvantagens podem ser mitigadas quando se usa RPCs que possuem mecanismos de linguagem de programação e de organização em hierarquia, ambos ausentes nas RPs. Tais características reduzem a complexidade do modelo, tornando-o mais compreensível e mais compacto, permitindo melhor visualização e entendimento. O emprego desses recursos encontra-se no escopo de nosso trabalho.

Na revisão bibliográfica, foi encontrado apenas um artigo com uma discussão completa sobre o uso de RPC para modelagem e visualização de um *game* completo. Trata-se do trabalho [10] que apresenta uma extensão da definição formal de RPC chamada de *game Coloured Petri Net* (GCPN). Quanto aos exemplos mostrados, esse trabalho apresenta apenas RPCs que modelam o tráfego de pacotes em uma rede de computadores. É feita apenas uma analogia entre a transmissão de pacotes e um *game* simplório tipo Ping-pong. Quanto à visualização do *game*, o autor mostra como seria possível usar a ferramenta SceneBeans [11] para visualizar o modelo de forma lúdica. Entretanto, não se trata de um caso geral. Para visualizações diferentes do exemplo de transmissão de

pacotes em uma rede, é necessária a implementação de uma interface em linguagem Java para customizar a visualização. Outro ponto que não é discutido no artigo é a jogabilidade, nada sobre como os jogadores irão interagir com o *game* é mencionado no artigo. Há muitas definições para o termo Jogos Sérios (JS). Entretanto, é consenso entre os autores da área que o termo se refere ao uso de jogos de computador que têm um propósito que não é apenas o entretenimento [12]. Os JSs têm ajudado a acelerar de maneira lúdica tarefas em diferentes áreas, tais como, negócios [13] [14], educação [15] [14] e saúde [12] [16]. Em [15] é apresentado um trabalho em que RPs são utilizadas em conjunto com ontologias para entender o processo de aprendizagem de um jogador. Cada ação de interesse pedagógico que o jogador utiliza é comparada com uma RP que mapeia as ações esperadas. Assim, se a ação executada pelo jogador tiver um caminho atingível na RP (sequência válida de disparos de transições), isso significa que ele escolheu um caminho de aprendizagem correto. Nessa abordagem, as ontologias representam o domínio de conceitos equivalentes. Desse modo, ações diferentes, mas semelhantes, podem ser analisadas com a ontologia, possibilitando reconhecer respostas equivalentes. Em termos de pesquisa de como softwares de propósito geral podem ser analisados dinamicamente, por meio de simulações e testes, ainda na fase de modelagem, os trabalhos [17] e [18] apresentam resultados relevantes. Em [17] são apresentadas heurísticas para transformar diagramas UML visualmente estáticos, tais como diagramas de classes e diagramas de sequência, em modelos de Redes de Petri Executáveis de Alto Nível (RPAN). Os conceitos de RPCs e RPANs são semelhantes e tratam do grau de abstração que as RPs podem ter com a introdução de código de alto nível, nas expressões de arco e nas condições de guarda. Em [18] RPANs são apresentadas como uma alternativa aos diagramas de *Business Process Execution Language* (BPEL), os quais são usados como modelos no paradigma de Arquitetura Orientada a Serviços (SOA). BPEL é bastante estática e várias abordagens têm sido propostas para permitir traduzi-la para um conjunto de formalismos diferentes. Desse modo, as RPANs são apresentadas como uma alternativa viável por permitir testes de modelos de forma interativa e dinâmica. De fato, os artigos [17] e [18] apresentam uma metodologia para transformar os diagramas visualmente estáticos da UML e BEPEL em RPAN que permite análise e testes dos softwares modelados ainda na fase de análise. Entretanto, nenhuma alusão é feita sobre a transformação das RPANs em software executável, de maneira a permitir a validação prévia dos requisitos funcionais pelos *stakeholders*.

## IV. Metodologia

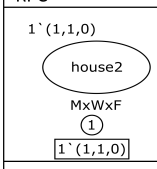
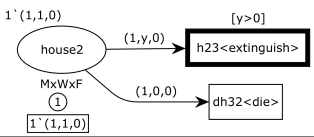
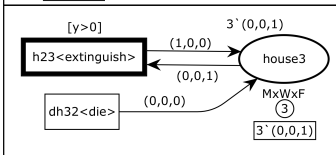
Como resultado deste trabalho, duas formas de utilizar RPCs no desenvolvimento de software sob uma abordagem *cornerstone* são apresentadas. A primeira técnica é voltada exclusivamente para *games* casuais. A segunda exemplifica como classes do paradigma de Programação Orientada a Objetos (POO) podem ser descritas, utilizando-se das RPCs para geração de uma interface gráfica destinada a validação e aos testes dos usuários finais.

### A. GAM – Videogames Assistidos por Modelagem

O método proposto, neste artigo, para construção de *games* fundamentados em RPCs é denominado de *Games* Baseado em Modelagem (GAM – *Game Aided by Modelling*). GAM constitui-se de um método de prototipagem *cornerstone* em que a implementação de *games* finais ocorre simultaneamente com análise, simulação e testes. RPCs permitem expressar a estrutura (implementação) e a dinâmica (comportamento) do *game* alvo, possibilitando fazer simulações e testes em fases precoces do desenvolvimento de software. Um estudo mais detalhado de como as RPs e as RPCs podem ser usadas para fins de análise e testes podem ser encontrados em [5] e [19]. Vale ressaltar que o conceito de GAM não está atrelado a nenhum *framework* de auxílio ao desenvolvimento de *games*, tais como, *Sprite Kit* [20], *Corona* [21] ou *Unity* [22] [23], sendo independente de plataforma. Para demonstrar como usar o método GAM, o *game* de um pequeno bombeiro preocupado em apagar as chamas em uma cidade, é apresentado a seguir.

Na Fig. 2, é mostrado um trecho de uma RPC para demonstrar como o método GAM pode abstrair os principais elementos de um *game*. Trata-se de um exemplo explicativo em que um bombeiro se desloca dentro de uma cidade para apagar focos de incêndio. Para representar os sprites, que são os elementos que se movimentam pelo mundo do *game*, foram utilizadas as fichas da RPC. O tipo de ficha product permite que se possa trabalhar com *n* dimensões na mesma ficha. Assim, foi utilizada uma ficha *product* com os seguintes campos (*Firefighter*, *Water*, *Fire*) para representar (bombeiro, água, fogo). Uma ficha (1,0,0) representa apenas um bombeiro, uma ficha (1,1,0) representa um bombeiro com água a sua disposição e uma ficha (0,0,1) representa apenas fogo. A Tabela I descreve a semântica dos elementos do modelo RPC da Fig. 2.

TABELA I
SIGNIFICADO DOS ELEMENTOS DA RPC DA FIG. 2 NO CONTEXTO DE *games*

| RPC | *Game* | Significado |
|---|---|---|
|  |  | O lugar *house2* significa uma região no mundo do *game*. A ficha 1`(1,1,0) significa um *sprite* bombeiro com uma unidade de água posicionada nessa região. |
|  |  | O lugar *house3* significa outra região no mundo do *game*. A ficha 3`(0,0,1) significa três unidades do *sprite* fogo nessa região. |
|  | | A transição *h23* será habilitada se existir um bombeiro com unidades de água e houver fogo em *house3*. Enquanto *dh32* será habilitada se o bombeiro não portar água. |
|  | | Se a transição *h23* for disparada, os fogos do lugar *house3* serão decrementados {ficha (0,0,1)}. Se o disparo ocorrer em *dh32*, o bombeiro morrerá (gera ficha (0,0,0)). |

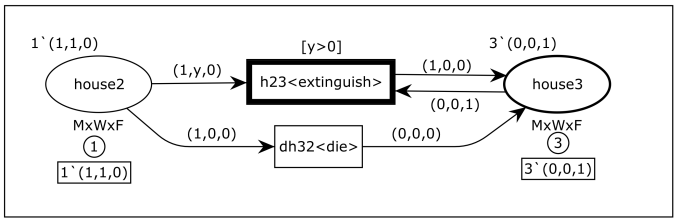



Fig. 2. Trecho de RPC para o *game* do bombeiro.

Observe que, em relação a um *game* casual qualquer, tanto a estrutura representada pelos lugares, quanto o comportamento representado pelas transições e arcos podem ser definidos por uma RPC. Assim, de posse do modelo do *game* em RPC e de posse de arquivos de imagem e de som, é possível desenvolver um protótipo interativo que pode ser usado na validação do *game* pelos usuários finais. Com mais refinamentos, tem-se um *game* casual completo. O software GAM Play que integra tais modelos ao desenvolvimento do *game* equivalente é apresentado na V-A.

### B. CAM – Classes Assistidas por Modelagem

De fato, o método GAM apresenta vantagens que possibilitam simulações e teses de um *game* ainda na fase de modelagem. Analogamente, tais benefícios podem ser obtidos para softwares de alto nível de propósito geral. O método denominado Classes Assistidas por Modelagem (CAM – *Classes Aided by Modelling*) utiliza-se de modelos de RPCs que podem expressar a estrutura (implementação) e a dinâmica (comportamento) de uma classe da POO. A primeira etapa para definição de uma RPC que possua os elementos de uma classe e que possa ser usada na POO consiste na definição de um lugar base, onde serão armazenadas as identidades dos objetos com seus respectivos atributos. No modelo RPC, apresentado na Fig. 3, é exposto um lugar chamado *forca* (força) associado a um conjunto de fichas coloridas do tipo *product* IDxMxA em que *ID* (identificador ou identidade) é do tipo *String*, *M* (massa) é do tipo *Int* e *A* (aceleração) é também do tipo *Int*. Percebe-se também que, na Fig. 3, foi adicionado um lugar chamado construtor com uma transição de mesmo nome. Esse arranjo permite que sejam instanciados objetos. Após os disparos da transição construtor, dois objetos são criados e armazenados no lugar forca: *id="f1", m=8, a=6*, *id="f2", m=3, a=4*. De modo a alterar o valor do atributo aceleração, foi inserido na RPC o lugar *setAceleracao* com sua respectiva transição de mesmo nome, conforme exposto na Fig. 3. Note que a ficha presente no lugar *setAceleracao* indica que o objeto de identidade *f1* receberá um parâmetro de aceleração de valor 10 (*pa=10*) após o disparo da transição *setAceleracao*. Neste momento, o objeto *f1* mudará seu estado de *id="f1", m=8, a=6* para *id="f1", m=8, a=10*. Essa dinâmica é descrita por meio das expressões presentes nos arcos de entrada e de saída da transição *setAceleracao*. Outros métodos como, por exemplo, *setMassa* são construídos da mesma maneira, sendo este suprimido por questão de economia de espaço para figuras e para escrita deste artigo. O processo de verificar um valor de um atributo de um objeto é exemplificado na Fig. 3 pelos arcos que ligam o lugar *getAceleracao* à transição de mesmo nome. Objetos podem informar valores calculados a partir dos atributos armazenados, ou seja, métodos com cálculos e que retornam valores. Um método para cálculo da força resultante (*f=m*a*) foi adicionado a RPC conforme exposto na Fig. 3. Tão logo a transição *calcForca* seja disparada, o cálculo da força é feito para os objetos presentes no lugar *calcForca*. O processo de seleção do objeto corrente e o cálculo da força são descritos pelos arcos de entrada e de saída da transição *calcForca*. Dessa forma, é proposta uma heurística para a construção de modelos RPC para especificação de classes do paradigma de orientação a objetos. Trabalhos como [17] e [18] propõem o uso de RPCs para facilitar análises e testes de softwares de alto nível, entretanto, a ferramenta CAM Play (veja V-C) transforma uma RPC em um protótipo executável, que pode ser manipulado por meio de interface gráfica. Dessa forma, os *stakeholders* já podem validar previamente as classes de domínio de um software.

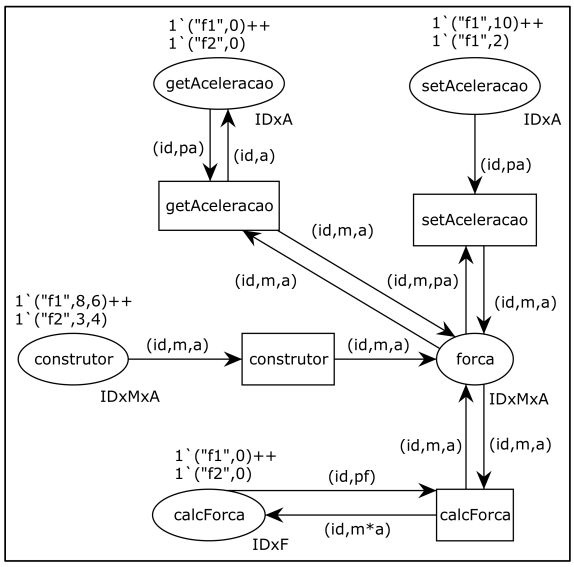



Fig. 3. Modelagem de uma classe Força (*f=ma*) com RPC.

## V. Resultados

Nesta seção, são apresentadas duas ferramentas de autoria própria para demonstrar como os métodos GAM e CAM (modelagem de *games* e classes respectivamente) podem ser implementados, de maneira a dar suporte visual aos softwares modelados com RPC, na seção IV de Metodologia. Também são discutidos como o uso de RPCs proporciona vantagens em relação a outras abordagens.

### A. GAM Play

Partindo-se dos modelos RPC desenvolvidos, analisados e validados na ferramenta CPN Tools [24], é possível gerar objetos Java a partir de uma RPC. O CPN Tools armazena suas RPCs em arquivos do tipo *.cpn, sendo estes codificados em XML (*Extensible Markup Language*) com *tags* específicas para os diversos tipos de elementos de uma RPC, tais como, lugar, transição, arco, ficha e conjunto de cores. Nota-se que esse tipo de arquivo contém os dados necessários para

transformar uma definição de uma RPC específica em uma hierarquia de classes da Linguagem Java. Na Fig. 4, é exibido um trecho do arquivo *.cpn que especifica uma RPC, mais precisamente, é exibido um componente do tipo lugar (*place*), evidenciando sua formatação em XML.

```
<place id="ID1412319740">
<posattr x="-395.000000" y="249.000000"/>...
<text>house1</text>...
<type id="ID1412319741">
<posattr x="-433.000000" y="275.000000"/>...
<text tool="CPN Tools" version="3.4.0">E</text>
</type>
<initmark id="ID1412319742">...
<text tool="CPN Tools" version="3.4.0">1`e</text>
</initmark>
</place>
```

Fig. 4. Exemplo de um trecho do arquivo *.cpn em formato XML, gerado pela ferramenta CPN Tools.

Basicamente, esse aplicativo tem como entrada uma RPC em formato XML gerada pelo CPN Tools (arquivo *.cpn) e arquivos de imagem e som. O GAM Play agrega uma camada de software que simula o ambiente do *game* em termos de movimentação dos *sprites*, colisões, animações e execução de sons. Dessa forma, esse aplicativo Java adiciona de maneira customizada as características multimídia e de controle dos personagens do *game*, enquanto a RPC especifica as características estruturais e de lógica comportamental. A seguir, são detalhadas a função das classes presentes na hierarquia exposta na Fig. 5

- *BaseGameEntity*: Movimentação linear e angular dos elementos de um *game*.
- *ImageEntity*: Manipulação de imagens e animações.
- *AnimatedSprite*: Abstração dos personagens que se movimentam no *game*, sendo subclasse de *Sprite*.
- *AnimationClass*: Execução do software tradutor de RPC para objetos Java propriamente ditos. Também trata a interface de interação do jogador com o *game*.
- *CpnXmlReader*: Provê dados para que a classe *ElementsCPN* possa definir objetos básicos que abstraem uma RPC.
- *ElementsCPN*: A partir de valores oriundos do arquivo *.cpn (XML), um objeto dessa classe instancia objetos das classes *Place*, *Transition*, *Arc* e *Token*. Em seguida, monta-se as matrizes *Pre* e *Post* estendidas para trabalharem com fichas coloridas. Assim, abstrai-se um espaço vetorial que controla os *sprites* conforme a especificação da RPC.

O GAM Play recebe como entrada a RPC e os arquivos de imagem e de som. Esses últimos estão associados aos nomes de lugares e de transições. Em seguida, cria o *game* automaticamente, ficando este totalmente independente do CPN Tools. O modelo RPC de uma fase completa do *game* do bombeiro é apresentado na Fig. 6. Por *default* os arcos sem

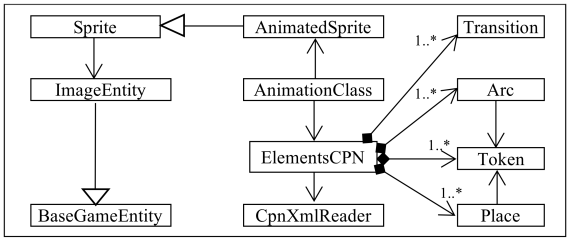



Fig. 5. Modelo Conceitual da Hierarquia de classes do software GAM Play.

inscrição recebem a expressão (*x,y,z*), a qual foi suprimida apenas para melhorar a legibilidade da figura.

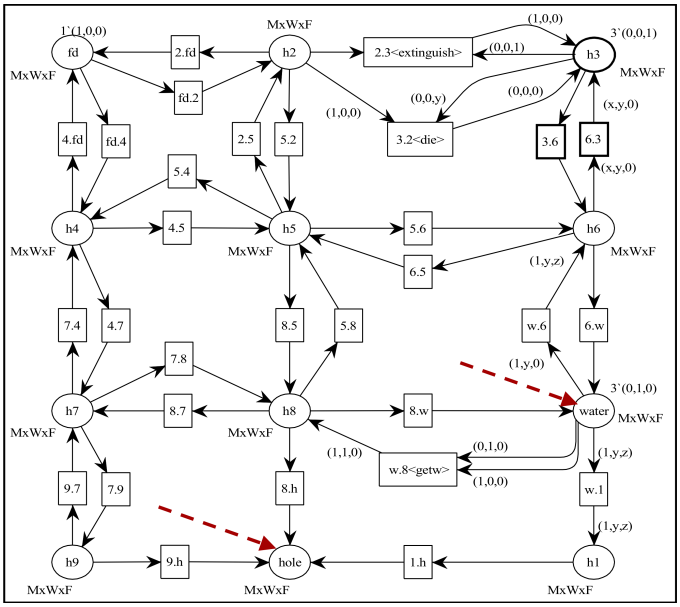



Fig. 6. Fase completa para o *game* do bombeiro representada por uma RPC.

Na Fig. 7.a, é apresentado o *game* interpretado pelo GAM Play após receber a RPC da Fig. 6, com os arquivos de imagem e de som associados aos lugares e às transições desta. A dinâmica de como o bombeiro adquire água pode ser visualizada no lugar *water* da Fig. 6. A forma como o bombeiro pode ficar preso em um buraco é modelada pelos sentidos dos arcos ao redor do lugar *hole* da mesma figura. Dessa maneira, é estabelecida uma dinâmica com a qual o jogador pode pegar água para apagar o fogo com cuidado para não se queimar nem cair no buraco. Por meio do GAM Play, inúmeros *games* podem ser gerados. Outro exemplo é exibido na Fig. 7.b, em que é apresentado um JS lúdico para o ensino da língua inglesa. Nesse jogo, uma caveira deve posicionar caixões rotulados com expressões idiomáticas da língua inglesa em uma cova de acordo com as preposições *in*, *on* e *at*. Parte do modelo RPC para esse JS é exibido na Fig. 8.

### B. Análise e Teste com CPN Tools

O mecanismo de análise de RPCs é uma ferramenta importante para verificar se os objetivos do *game* podem ser alcançados, mesmo antes do GAM Play ser usado para gerar um *game*. Usando a ferramenta de análise de espaço de estados

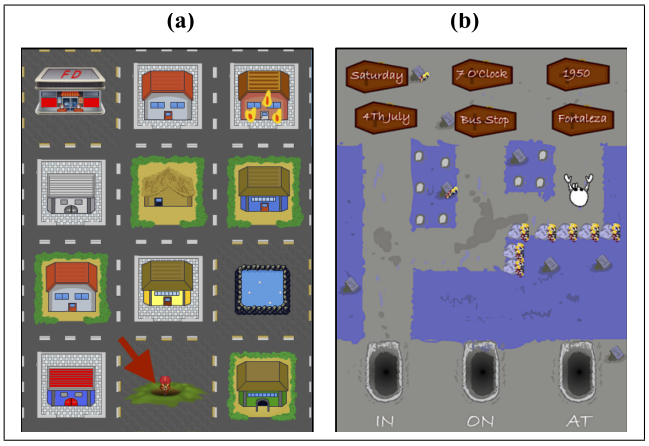



Fig. 7. a) Fase completa de um *game* construído com a metodologia GAM a partir da RPC da Fig. 6, evidenciando o bombeiro preso no buraco. b) exemplo de um *game* educacional lúdico também desenvolvido pela metodologia GAM, em que os elementos *saturday, 4th July* e *IN* foram modelados na Fig. 8.

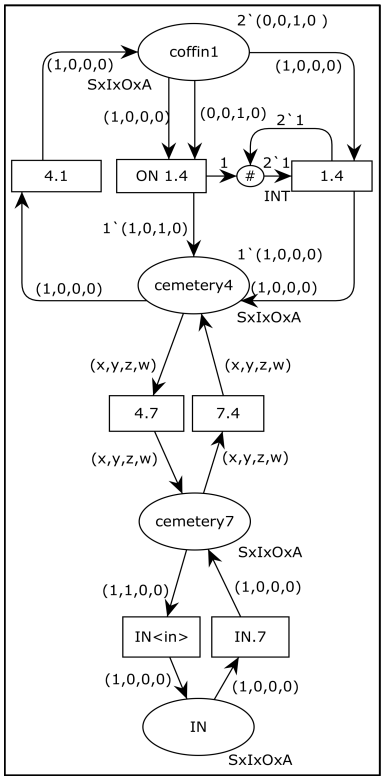



Fig. 8. Modelo simplificado de RPC para um JS para o ensino do uso correto de preposições da língua inglesa. Para melhorar a visualização, foi exibida apenas a primeira coluna da RPC que mapeia os elementos *saturday*, *4th July* e *IN* da Fig. 7.b.

do CPN Tools, foi feita a análise da PRC da Fig. 6. Assim, encontrou-se o seguinte resultado:

- A rede possui 77 marcações alcançáveis a partir da marcação inicial significando neste caso, 77 jogadas ou movimentações possíveis para o bombeiro.
- A rede possui 10 marcações mortas, ou seja, 10 marcações que não possuem nenhuma transição habilitada. Isso significa que há 10 maneiras do *game* acabar, seja com vitória, seja com derrota. Assim, foi possível identificar as possibilidades de: queda do bombeiro no buraco, tentativa de apagar o fogo sem água e apagar o fogo com sucesso.

A título de exemplo, na Fig. 9, é apresentada uma sequência de marcações geradas pelo CPN Tools que leva o bombeiro a apagar o fogo completamente. Observa-se, nessa figura, que resta apenas a ficha 1‘(1,0,0) no lugar *bomb’house3* (mesmo que *h3* na Fig. 6), o que significa que não há mais fichas representando água ou fogo (situação de vitória). Note que, embora só tenham sido apresentados os disparos das transições que levam o bombeiro à vitória, em quase todas as marcações apresentadas existe mais de uma transição habilitada, o que poderia levar o bombeiro a uma situação de falha em seu objetivo. Por exemplo, a marcação 54 apresentada na Fig. 9 (penúltima marcação) possui duas marcações antecessoras e duas marcações sucessoras, o que é evidenciado pela informação ”2:2”, explicitado na referida marcação. Nesse caso, pod eria ter sido disparada uma outra transição quando o bombeiro alcançou essa marcação e, dessa forma, o estado desejado (61) não teria sido alcançado.

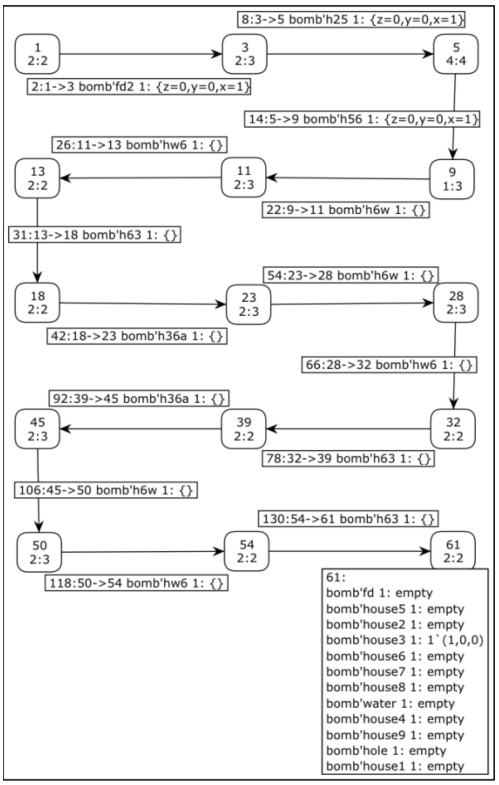



Fig. 9. Exemplo de sequência de disparos de transições que leva o bombeiro a exterminar o fogo, portanto, com vitória para o jogador.

### C. CAM Play

O CAM Play é uma ferramenta semelhante ao GAM Play que utiliza o processo de interpretação para gerar um software final conforme uma especificação de classe feita com RPCs. O protótipo gerado pode ser usado para testes funcionais e validação final por parte dos *stakesholders*. Na Fig. 10, há um exemplo da interface gráfica do CAM Play para o cálculo

do valor da força do objeto *f1* da Fig. 3. Dessa maneira, há possibilidade de uso do protótipo em fases preliminares à codificação de software. O CAM Play ainda pode ser integrado com outros softwares, de modo que a classe modelada possa ser usada para outros fins, permitindo seu reuso e aumentando a produtividade.

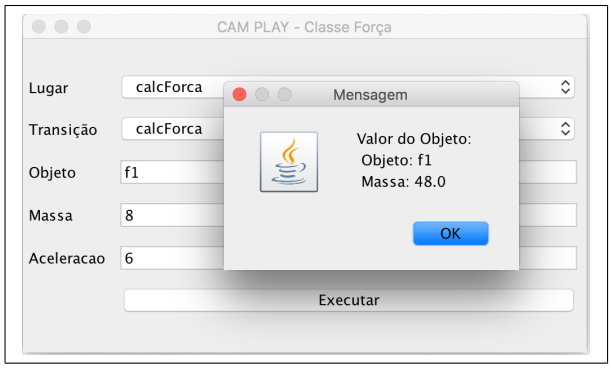



Fig. 10. Interface gráfica do CAM Play exemplificando o cálculo da força para o objeto *f1*.

### D. Discussões

A indústria de *games* já rivaliza com a indústria do cinema. *games* complexos requerem centenas de profissionais para desenvolvê-los. Assim, propor mecanismos de geração automática de *games* complexos é uma tarefa ambiciosa. Entretanto, *games* casuais de diversos tipos podem ser facilmente desenvolvidos, segundo o método GAM, tal qual foi demonstrado com o jogo do bombeiro (Fig. 6 e Fig. 7.a). Uma das questões fundamentais no projeto de um *game* é saber se o público-alvo realmente tem empatia com os personagens, as músicas e a jogabilidade (*playability*). Demonstrar essas características para uma parcela do público-alvo em estágios precoces do desenvolvimento do *game* pode ser custoso. O método GAM apresenta uma solução eficaz para esse problema, pois como foi demonstrado a prototipação e mesmo o *game* final podem ser rapidamente gerados. A jogabilidade e o estilo do *game* modelado dependerão da ferramenta que transforma a RPC em *game*. No caso deste artigo, foi proposto o GAM Play. Como foi demostrado nos exemplos de *games* deste artigo, o GAM Play pode gerar diversos *games* casuais diferentes do tipo aventura, inclusive com mecanismos de jogabilidade transparentes para o programador. Com a evolução do GAM Play, outros estilos de *games* também serão possíveis: corrida, RPG e plataforma. Outra questão importante no desenvolvimento de *games* é o número de fases disponíveis em um jogo. Um dos itens de sucesso para um *game* é um número de fases em que o jogador possa se divertir, se possível, por vários meses. O método GAM também pode ser empregado nesse aspecto, visto que o mecanismo de simulação do CPN Tools faz com que o projetista possa visualizar a dinâmica da fase, inclusive com o movimento dos sprites. Quanto ao questionamento de uma possível explosão de estados, o uso da hierarquia provê modularização e o reuso de RPCs. Inclusive, uma solução já foi demonstrada em outros trabalhos de nossa autoria [25] e [26]. Com o mecanismo de hierarquia, um modelo RPC pode ser criado de maneira modular, com unidades de RPCs independentes e interconectadas. Essas unidades podem modelar os vários níveis de abstração. Além disso, outra maneira de diminuir a complexidade é o uso da linguagem CPN ML. No jogo do bombeiro, as fichas do tipo product permitem que os *sprites* bombeiro, água e fogo possam ser posicionados no mesmo lugar ao mesmo tempo. Inclusive, é possível a execução de movimentos assíncronos, permitindo que personagens distintos atuem paralelamente em diferentes lugares. No jogo da caveira, o próprio personagem e as expressões idiomáticas relacionadas com *in*, *on*, *at* também não possuem restrições de posicionamento. Assim, com o uso de fichas do tipo *product*, *n* objetos diferentes podem interagir em um único lugar de uma RPC ao mesmo tempo. Em uma RP simples, seria necessário *n* lugares para receber *n* objetos distintos, tornando o modelo mais complexo. Dessa forma, não foram apenas expostos os benefícios inerentes às RPCs, mas também como aplicar tais benefícios no contexto de desenvolvimento de *games*. Esse fato também pode ser dito em termos de testes de modelo, por exemplo, as explicações da Fig. 9. O diferencial do método CAM em relação ao diagrama de classes da UML é que o modelo proposto em RPC é dinâmico e as ferramentas de desenvolvimento de RPC propostas podem apresentar visualmente os estados de um objeto, além da possibilidade de gerar um protótipo final para interação com os *stakeholders*. Assim, por apresentar um protótipo visual, o método CAM apresenta benefícios a mais que as propostas apresentadas em [17] e [18]. Por questões didáticas, para demonstrar o uso do método CAM, escolheu-se uma classe em que a principal operação matemática é $f=m*a$. Entretanto, vale ressaltar que o uso da linguagem CPN ML possibilita a construção de métodos mais complexos e de propósitos gerais.

## VI. Conclusões e Trabalhos Futuros

As RPCs permitem que os processos de modelagem e implementação de software ocorram de forma evolucionária, em que os refinamentos dos modelos possam, com auxílio de ferramentas, gerar o software alvo automaticamente. Desse modo, análises, prototipação e testes podem ser realizados desde a fase inicial do desenvolvimento de maneira ágil, sem gerar sobrecarga de trabalho e diminuído o esforço na escrita do código do software. O CPN Tools, por exemplo, permite que tais análises e testes sejam feitos sob os referidos modelos. Esses mesmos modelos servirão de entradas para ferramentas de geração do software final, tais como, GAM Play e CAM Play. De fato, tais ferramentas podem agregar produtividade no desenvolvimento de *games* e de classes de linguagens de POO. Dessa forma, as vantagens da modelagem e da prototipação de software podem ser melhor percebidas pelos desenvolvedores, deixando de ser negligenciadas em nome de uma suposta economia de trabalho. A próxima etapa na evolução do GAM Play é prover uma ferramenta para substituir fichas e expressões da linguagem CPN ML por ícones de maneira que usuários leigos em RPC possam desenvolver *games*. Uma proposta de evolução da ferramenta CAM Play é a introdução de um mecanismo de compilação em

substituição ao método de interpretação atual. Além de permitir um melhor desempenho do software final, manutenções ou evoluções serão facilitadas, pois os programadores poderão alterar o código mais facilmente. Ferramentas de compilação de RPC semelhantes já foram construídas para software de baixo nível conforme nossa contribuição exposta em [27], em que foi criado um processo de tradução por compilação de RPCs para diagramas da linguagem Ladder.

## REFERÊNCIAS


[1] J. R. G. Booch and I. Jacobson, *The Unified Modeling Language User Guide*, 1st ed. Rio de Janeiro: Addison Wesley, 1998.

[2] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[3] M. Hauder, S. Roth, C. Schulz, and F. Matthes, "Agile enterprise architecture management: an analysis on the application of agile principles," in *International Symposium on Business Modeling and Software Design BMSD*, 2014.

[4] R. Wazlawick, *Engenharia de software: conceitos e práticas*. Elsevier Brasil, 2013, vol. 1.

[5] K. Jensen and L. M. Kristensen, *Coloured Petri nets: modelling and validation of concurrent systems*. Springer Science & Business Media, 2009.

[6] J. D. Ullman, *Elements of ML programming*, 2nd ed. Prentice-Hall, 1997.

[7] Y.-S. Lee and S.-B. Cho, "Context-aware petri net for dynamic procedural content generation in role-playing game," *IEEE Computational Intelligence Magazine*, vol. 6, no. 2, pp. 16–25, 2011.

[8] ——, "Dynamic quest plot generation using petri net planning," in *Proceedings of the Workshop at SIGGRAPH Asia*. ACM, 2012, pp. 47–52.

[9] M. Araújo and L. Roque, "Modeling games with petri nets," *Breaking New Ground: Innovation in Games, Play, Practice and Theory. DIGRA2009. Londres, Royaume Uni*, 2009.

[10] M. Westergaard, "Game coloured petri nets," in *Proc. of 7th CPN Workshop*, vol. 579, 2006, pp. 281–300.

[11] N. Pryce and J. Magee, "Scenebeans: Acomponent-based animation framework for java," Citeseer, 2001.

[12] P. Rego, P. M. Moreira, and L. P. Reis, "Serious games for rehabilitation: A survey and a classification towards a taxonomy," in *5th Iberian Conference on Information Systems and Technologies*. IEEE, 2010, pp. 1–6.

[13] R. J. Rossetti, J. E. Almeida, Z. Kokkinogenis, and J. Gonçalves, "Playing transportation seriously: Applications of serious games to artificial transportation systems," *IEEE Intelligent Systems*, no. 4, pp. 107–112, 2013.

[14] A. B. Saavedra, F. J. A. Rodriguez, J. M. Arteaga, R. S. Salgado, C. A. C. Ordonez, and J. A. H. Alegria, "Verification and validation model for short serious game production," *IEEE Latin America Transactions*, vol. 14, no. 4, pp. 2007–2012, 2016.

[15] P. Thomas, A. Yessad, and J.-M. Labat, "Petri nets and ontologies: Tools for the" learning player" assessment in serious games," in *Advanced Learning Technologies (ICALT), 2011 11th IEEE International Conference on*. IEEE, 2011, pp. 415–419.

[16] S. Cano, J. M. Arteaga, C. A. Collazos, C. S. Gonzalez, and S. Zapata, "Toward a methodology for serious games design for children with auditory impairments," *IEEE Latin America Transactions*, vol. 14, no. 5, pp. 2511–2521, 2016.

[17] A. Alhroob, K. Dahal, and A. Hossain, "Transforming uml sequence diagram to high level petri net," in *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, vol. 1. IEEE, 2010, pp. 260–264.

[18] J. Fabra and P. Álvarez, "Bpel2deneb: translation of bpel processes to executable high-level petri nets," in *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*. IEEE, 2010, pp. 496–505.

[19] M. Tadao, "Petri nets: properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, 1990.

[20] A. Ubillis, "Evaluation of sprite kit for ios game development," Master's thesis, Department of Computer and Information Science, Human-Centered systems, Linköping University, Linköping, Nov. 2014.

[21] M. P. A. Balayan, V. V. B. Conoza, J. M. M. Tolentino, R. C. Solamo, and R. P. Feria, "On evaluating skillville: An educational mobile game on visual perception skills," in *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*. IEEE, 2014, pp. 69–74.

[22] F. A. Pires, W. M. Santos, K. d. O. Andrade, G. A. Caurin, and A. A. Siqueira, "Robotic platform for telerehabilitation studies based on unity game engine," in *Serious Games and Applications for Health (SeGAH), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 1–6.

[23] H. Zhong and J. Xiao, "Apply technology acceptance model with big data analytics and unity game engine," in *Software Engineering and Service Science (ICSESS), 2015 6th IEEE International Conference on*. IEEE, 2015, pp. 19–24.

[24] F. Yutao, S. Guiping, H. Liu, and Z. Siyu, "Study on a cpn-based auto-analysis tool for security protocols," in *Information Science and Engineering (ISISE), 2012 International Symposium on*. IEEE, 2012, pp. 179–182.

[25] V. V. S. Carvalho, C. H. R. Gonçalves, J. C. Soares, F. M. Barreto, J. M. Soares, and G. C. Barroso, "A formal method to generate games with multi-level and multi-user using hierarchical coloured petri nets," in *13th Brazilian Symp. in Computer Games and Digital Entertainment, Porto Alegre*. SBC, 2014.

[26] ——, "Criação de jogos didáticos de perguntas e respostas por meio de modelagem em redes de petri coloridas hierárquicas - development of questions and answers for didactic games through modeling with hierarchical colored petri nets," in *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 25, no. 1. SBC, 2014, p. 352.

[27] J. R. Costa, "Algoritmo de conversão de redes de petri coloridas para ladder logic diagram (lld)," Master's thesis, Dep. Teleinformática, Uni. Federal do Ceará, Fortaleza, Jan. 2014.


**Carlos Hairon Ribeiro Gonçalves** holds a Master's Degree in Computer Science from the Federal University of Ceará (2003). He is professor in the Department of Telematics in Federal Institute of Education, Science, and Technology of Ceará (IFCE). He has experience in Computer Science, working mainly: Internet, Mobile Devices, Information Systems, and Intelligent Systems.

**José Marques Soares** holds a Doctorate from Réseaux, Connaissances et Organisations by the Institut National de Télécommunications (2004). He is currently an adjunct professor in the Department of Teleinformatics Engineering at the Federal University of Ceará (UFC). He has experience in Computer Science, working mainly: Design and Development of Applications to Support Education, Modeling, and Construction of Simulators with Colored Petri Nets and Identification of Human Postures by Computer Vision.

**Giovanni Cordeiro Barroso** holds a doctorate in Electrical Engineering from the Federal University of Paraíba (1996). He is currently a titular professor in the Department of Physics at the Federal University of Ceará (UFC). He has experience in Electrical Engineering and distance education, working mainly: Supervisory Control, Colored Petri Nets, Smart Grids, Evolutionary Algorithms, Analysis and Modeling of Productive Systems and Evaluation in Distance Education.