Debate de Estrutura do TP 2

Introdução

Tendo em uso constate os predicados da GLib, serve o presente documento como metódo de apresentação de uma estrutura preliminar para a estrutura de dados encarregue por todos os comportamentos deste mecanimo.

Pretende-se que o *output* final seja o de uma página HTML com todas as relações necessárias. Como tal, e à semelhança do que foi feito no TP1, é imperioso possuir uma estrutura de dados principal de gere todos os diferentes tipos de informação, e que no final fica encarregue, seguindo os conectores adequados, de produzir o resultado desejado.

Dessa forma, o que se pretende é encontrar o que é que pode vir a ser a estrutura que possua uma maior flexibilidade na explicitação de todas as componentes do sistema.

Componentes Atómicas

Antes de qualquer debate é necessário considerar que este sistema visa definir unidade atómicas de atuação, nomeadamente, incluir as seguintes temáticas:

- **Caderno** é o ponto de acesso base, o *start point*, da gramática independente de contexto, é a unidade mais abstrata de todas. Um caderno é composto 0 ou mais pares de <1 documento, 0 ou mais triplos>.
- **Documento** é a unidade que define, de certa forma, o assunto a ser retratado. O documento é composto pelo tuplo , sendo que todos estes retratam componentes úteis do ponto de vista descritivo da própria entidade em questão.
- **Triplo** é a unidade relacionamentos base de toda esta mecânica, sendo composta pelo tuplo , indicado qual é a relação entre um sujeito e um objecto. Sendo que ambos o sujeito e objecto representam tópicos principais.

Resultado Desejado e como o obter

O interesse máximo desta aplicação é conseguir gerar páginas HTML que agrupem informação associada a cada um dos conceitos e que "hiperlink com tudo o que for possível", claro que esta é uma noção imensamente vaga, e que os professores, evidentemente, procuram que nos aprofundemos. Acho que também faz sentido que exista páginas HTML só para as relações em si, e desta forma sendo possível obter todos os conceitos que estão envolvidos entre si segundo uma dada relação.

Seria também imensamente interessante conseguir desenvolver uma especie de grafo que vise representar estas entidades subjacentes. Pois tudo isto trata-se apenas de um grafo, em que os arcos são as relações entre os diferentes conceitos, sendo estes os respectivos nós.

Estas ideia analogia direta com grafos, permite imeditamente fazer uma tradução direta para o mecanismo pretendido. Ora, quando temos um nó, então poderemos vir a ter um numero exponencial de arcos a sair deste mesmo. E, simetricamente, um número exponencial de arcos a chegar ao nó destino. Desta forma, podera ser vantajoso representar os conceitos como sendo entradas, ou biprodutos, de uma tabela de hash.

Tendo em conta os resultados do anterior documento preliminar, é notável que as relaçes possuiram uma forte componente no âmbito dos metadados, e que uma arquitetura em torno das relações, em oposto a uma arquitetura baseada nos conceitos, poderá ser mais vatajosa no que toca ao estabelicemento e manejo de diferentes relações e as interligações entre si.

Porém, ao optarmos pela arquitetura em torno das relações, perdemos a capacidade de referenciar conceitos unívocamente, o que arrecata ainda mais a perder, tendo em conta que se pertende definir triplos, em que o sujeito é um conceito. Denotando, desde sempre, a sua exterma importância.

Como tal, poderá vir a ser de extremo interesse tentar obter uma arquitetura não compremetedora, que constituia um balanço entre a definição de relações e a definição de conceito, idealmente, em tempo assimptótico constante.

Precisamos portanto de um mecanismo capaz de interligar, possivelmente por meio de apontadores partilhados, os dados armazenados em ambas as tabelas, de forma a oferecer um compromisso de eficácia entre as duas entidades.

Estudo Aprofundado das Relações

Com mente em todos os extras que decidimos adicionar, maioritariamente, todos no âmbito das relações. Temos de ter um especial cuidado ao pesar as diferentes dinâmicas entre relações.

Primeiramente, sublinha-se que há necessariamente a capacidade de introduzir relações reservadas, relações estas que possuem um funcionamento bem definido. Como tal, tenhamos em atenção algumas destas.

Relação `img`

é uma relação em que o sujeito fica associado a um objecto, sendo que este objecto representa uma imagem que será, posteriormente, associada ao sujeito na página HTML. Imeditamente surge um problema que necessitamos de enfrentar imediatamente, devemos permitir que mais de uma imagem seja associada a um

mesmo sujeito? a resposta é sim, e num cenário real faz todo o sentido que eu queria adicionar ao sujeito "familia" várias fotografias de diferentes membros do meu agregado familiar.

Estas fotografias devem ser posteriormente inseridas num slideshow de fotografias adequado à temática, de forma a garantir que todas as fotografias associadas são apresentadas de seguida. Isto traz outro ponto, assim sendo, será que devemos permitir a inserção de imagens repetidas? Por exemplo, será que faz sentido adicionar ao album da familia 2 fotos identicas de um dos membros? A resposta é, intuitivamente, não faz muito sentido, mas não podemos assumir o objetivo do utilizador da ferramenta.

Pode ser do interesse do utilizador intercalar fotos repetidas para provar um ponto, caso no qual a nossa ferramente falharia redondamente. Tendo isto em mente, penso que o mais adequado seria aceitar imagens repetidas. Porém, avisar o utilizador, por via de terminal, sempre que um caso de relações repetidas ocorra.

Mas então, de que forma intercetamos um destes casos? A resposta é simples, sempre que tivermos de efetuar uma inserção na tabela de hash das relações, basta-nos verificar se esse elemento já não está lá inserido. Se tiver, é emitido um aviso, e a insersação é efetuada de qualquer das formas. Porém, isto traz problemas adicionais, porque então temos de ter em atenção de que forma descrevemos o valor associado a cada relação, não pode simplesmente ser um array, caso contrário estariamos a ir contra todos os padrões de eficácia.

Na verdade pode de facto ser um array, ao qual simplesmente damos append de todos os valores que recebemos, assumindo que no futuro, eventualmente, estes dados serão processados e uma mensagem de *warning* deverá emergir aquando deste momento. Isto permite uma estrutura simples e pouco dispendioso, tirando mais partido do CPU para delinear futuras correspondências.

Relação `attach`

Esta relação é em todos os aspectos identica à relação debatida no ponto acima, apresentando, por essa razão, um problema de igual facilidade de resolução.

Relação `inserveOf`

Esta é uma relação reservada que ainda possui alguma complicações. Tenhamos o exemplo de `:filho_de :inverseOf :pai_de`, na prática isto significa que ser filho é o inverso de ser pai de alguém, porém o simétrico é igualmente válido, ser pai de alguém é o inverso de ser filho de alguém, desta forma, surge uma evidente relação bidirecional.

Segundo a tabela de relações que indicamos acima, vamos assumir que abaixo da relação `filho_de` possuimos o par que indica que X é filho de Y, então assumindo

que `pai_de` é o seu inverso, teremos, necessariamente, abaixo de `pai_de` um tuplo , representativo de Y é o pai de X.

Este é dos poucos casos em que as relações se aplicam também elas próprias sobre relações, e isto só pode acontecer em casos estritos e reservados. Quando isto acontece, estes casos devem ser lidados em pós-processamento, construindo uma tabela de hash, que possui só relações reservadas ao nivels dos metadados, como `inverseOf` e `synonym`. E que permite definir uma transição, usando funções de ordem superior, entre estas duas componentes. Todas as componentes de uma relação de uma das tabelas deverão ser copiadas, com os seus valores trocados, para a outra tabela, e vice-versa. Este mecanismo ainda tem muita coisa que se lhe diga, e o seu comportamento pode não ser, de todo, tão linear como se identifica aqui.

Relação `synonym`

É uma relação em tudo igual, exceto num funcionamento, â relação de cima. De forma a simplificar, contraria-se o documento anterior, no qual se indicava que esta relação poderia afetar tanto conceito como relações. Devido a questões de simplicidade, iremos assumir que esta relação afeta unicamente relações.

Como tal, o resultado final corresponderá a copiar os valores entre tabelas de relações que são sinonimas, sem trocar nada (ao contrário do que acontecia no predicado `inverseOf`).

Relação Abstrata

Futuramente, aquando da implementação, deveremos ter presente que qualquer mecanismo que permita um *dynamic dispatch* de funções de ordem superior para lidar com relações reservadas é de todo o interesse, tanto do ponto de vista de escrita de código, como do ponto de vista de escabilidade do sistema de forma a acautelas potenciais novas relações reservadas.

Conclusão

Por fim, conclui-se que qualquer que venha a ser a estrutura a ser utilizada, deverá sempre ter um enorme foco nas relações, diferenciado entre relações normais e relações reservadas. Partindo das relações subjacentes, somos capazes de alcançar todo a informação, para já, necessária às restantes componentes.

Resulta do estudo um forte recomendação para o estabelecimento de uma estrutura composta por duas tabelas de hash, uma para armazenar predicados normais e outra para armazenar relações reservadas. Sendo que a cada uma destas estará associado um growing array contendo tuplos de sujeito e objecto de forma a indicar os intervenientes em cada dada relação em tempo constante.

Devido a esta organização semi-estruturada, é óbvio que deveremos incluir mais poder de processamento, por meios do CPU. Porém, não achamos que isto sera um grande entreve, assumindo um utilização inteligente destes mecanismos, especialmente no âmbito de tabelas de hash, o tempo de construção e impressão de todos os elementos devidamente formatados continuara a ser, assimptoticamente, linear.