REFERÊNCIAS BIBLIOGRÁFICAS

1. BOOCH, G; RUMBAUGH, J e JACOBSON, I: *UML, Guia do Usuário: tradução*; Fábio Freitas da Silva, Rio de Janeiro, Campus ,2000.

2. BREUKER, J.A. and WIELINGA, B.J. *Interpretation of Verbal Data for Knowledge Acquisition. Advances in Artificial Intelligence*. T. O'SHEA Ed., Amsterdam, North Holland, 1987.

3. BREUKER, J.A. and WIELINGA, B.J. *Models of Expertise in Knowledge Acquisition. Topics in Expert System Design: Methodologies and Tools* . Guida G. and Tasso C. Eds., Amsterdam, North-Holland, 1988.

4. FURLAN, J. D. *Modelagem de Objetos Através da UML*: São Paulo, Brasil, Makron Books, 1998.

5. GIARRATANO, J. C. and RILEY, G. *Expert Systems: Principles and Programming. Boston: PWS-KENT Publishing Company*, 1989.

6. GOTTGTROY, M. P. B. O *Processo de Aquisição do Conhecimento na Construção de Sistemas Especialistas*: Rio de Janeiro. M. Sc. Dissertação COPPE Universidade Federal do Rio de Janeiro, 1990.

7. GOTTGTROY, P. C. M. *Uma Proposta de Modelo Conceitual para a Elicitação de Requisitos de Sistemas Dinâmicos*: Dissertação (Mestrado em Sistemas e Computação) DIMAp/CCET/UFRN, Natal, 2000.

8. de GREEF, P. and BREUKER, J.A. *A Case Study in Structural Knowledge Acquisition*. Int. Joint Conferences on Artificial Intelligence, august, Los Angeles, California, 1985. pp. 390-392.

9. KASABOV, N. K. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering: Massachusetts Institute of Technology Press, 1989*.

10. LARMAN, G. *Utilizando UML e padrões: Uma introdução à análise e ao projeto orientados a objetos*; Tradução Luiz A Meirelles Salgado. Bookman Porto Alegre, 2000.

11. de OLIVEIRA, M. A. and GOTTGTROY, M.P.B. Multi-Agents Conception for Implementation of the Approximate Reasoning. In *CIT'99: Trend in Information Technology*, MOHANTY H. and BARAL C. Eds., Tata McGraw Hill, 2000. pp 115-120.

12. RUMBAUGH, J. *Modelagem e Projetos Baseados em Objetos*: Rio de Janeiro: Editora Campus, 1994.

13. RUMBAUGH, J. *Models Through the Development Process. Journal of Object-Oriented Programming*. Maio de 1997. NY,NY: SIGS *Publications*.

14. SCHREIBER, G. Et all. Knowlede Engineering and Management: The CommonKADS Methodology. MIT Press, 1999.

15. SILVA, L. F. *Modelagem Conceitual como Ferramenta para o Desenvolvimento de Sistemas Computacionais*: Monografia do Curso Bacharelado em Ciências da Computação, DIMAp-UFRN, Natal, 1999.

16. SIQUEIRA, A.C.L. *Estudo Conceitual do Paradigma Baseado em Agentes como Solução Computacional de Problemas*: Dissertação (Mestrado em Sistemas e Computação), DIMAp/CCET/UFRN, Natal, 2000.

17. SOUZA, L.M.M. *Metodologias para Desenvolvimento de Sistemas Dinâmicos*: Dissertação (Mestrado em Sistema e Computação), - DIMAp/CCET/UFRN , Natal, 1999.

18. SPENCER, C. *LPA Prolog in Action.* In PC AI, November/December 1997. pp. 40-42.

O *framework* descreve os modelos que serão utilizados para simplificar a realidade e ajudar na construção de sistemas. Esses modelos visam fornecer uma cópia do sistema, onde podem abranger planos detalhados e gerais, dando uma visão panorâmica do sistema. Um bom modelo inclui aqueles componentes menores, que não são relevantes ao sistema em determinado nível de abstração. Segundo a visão da UML todos os sistemas podem ser descritos sob diferentes aspectos, com utilização de modelos distintos. Na sua visão cada modelo terá, portanto, uma abstração semanticamente específica do sistema. O objetivo principal dos modelos é compreendermos melhor o sistema que estamos construindo. A modelagem do sistema deve atingir cinco objetivos: primeiro um entendimento do entorno do problema; segundo ajudar na visualização do sistema tanto considerando para que ele se destina como a forma que está sendo construído e esperamos que ele seja implementado. O terceiro objetivo é permitir especificar a estrutura ou o comportamento de um sistema. O quarto objetivo é documentar as decisões tomadas. O quinto objetivo é proporcionar um caminho para a construção do sistema.

O *framework* não só utiliza modelos para desenvolver grandes sistemas, mas também auxilia na construção de pequenos sistemas. Sendo que nos grandes sistemas sua importância é maior, já que auxilia na compreensão de sua totalidade, antes dos mesmos serem divididos. Particionam-se os sistemas devido ao limite da capacidade humana de compreender a sua complexidade. Com ajuda dos modelos, delimitamos o problema que está sendo analisado, restringindo o foco a um único aspecto por vez, de forma a poder soluciona-lo.

O método proposto pelo o *framework* define que seguindo esse caminho, ou seja, começando a construção do sistema com a elicitação do conhecimento, utilizando a técnica adequada de aquisição do conhecimento, e depois fazendo a modelagem conceitual qualitativa. Conseguimos obter um modelo conceitual que identifica conceitos, (objetos) no domínio do problema. A utilização agora dos diagramas de caso de uso terá um enfoque mais especifico, onde serão atribuídas as responsabilidades necessárias para se obter visões particionadas do sistema já que o mesmo segue o conceito de análise de requisito. Os diagramas de atividade, e se necessário os diagramas de interações darão a visão das interações que do sistema necessita e com o diagrama de classe já poderemos começar a implementar.

Quando o desenvolvedor chegar no nível lógico do sistema, ele já tem noção do conhecimento que foi perdido no transcorrer da construção do sistema. Nesse caso ele já pode definir como as regras de produção poderão absorver o conhecimento perdido e como ele poderá auxiliar na lógica da base de conhecimento do sistema. Nessa fase o analista já tem um mapeamento genérico dos relacionamentos, que de forma clara já pode definir como irá implementar o sistema. Todo o desenvolvimento do *framework*, vai seguir por cada fase do ciclo de vida de acordo com a trajetória dos princípios da metodologia KADS, onde será conduzido de forma dinâmica, definindo o limite que existe entre a automação e a linha imaginária de abstração na construção de um sistema, assim como, não vamos deixar de considerar o limite que é imposto pelo sistema computacional na fase de implementação.

A linguagem de modelagem da UML apresenta um pouco de limitação em seus métodos, já que trabalha apenas parcialmente o nível conceitual, pois apesar dos autores da UML considerar o modelo conceitual desassociado de qualquer mecanismo lógico de implementação, o modelo em si difere pouco dos modelos lógicos usados por eles na construção de sistemas. O fato da UML não ter sido vinculada a um processo, tem dificultado a construção de sistemas, por não definir um processo que sirva de guia para construção de sistemas, e os métodos que eles consideram como PMR, tem um ciclo de desenvolvimento complexo de se trabalhar por não decompor totalmente, os modelos no nível conceitual e não ter a visão real do escopo do domínio, condição necessária para análise de requisitos. Como conseqüência disso termina sendo que a UML que hoje é uma linguagem de modelagem consolidada no mercado, sendo usada inadequadamente por conseqüência de não ter sido vinculada a um processo que definisse o processo de análise e ajudasse na construção do sistema.

Um trabalho futuro é desenvolver um sistema que possibilite escrever cada um dos modelos usados no *framework*, junto com um mecanismo que permita realizar uma verificação automática nas passagens de uma etapa (modelo) a outro.

CONCLUSÕES

Neste trabalho foi proposto um *framework* para desenvolvimento de sistemas complexos, isto é, sistemas que exijam muitos requisitos computacionais devido à diversidade das informações. Esses problemas se multiplicam com o desenvolvimento acelerado da tecnologia e necessitam de soluções rápidas que facilitem a vida moderna. Eles envolvem requisitos que geram alta complexidade, tanto para sua descrição como para sua solução computacional. Hoje analisar os requisitos e a dinamicidade de um sistema é um dos maiores desafios para os desenvolvedores. Já que eles necessitam representar a evolução dos conceitos em processos dinâmicos.

O *framework* proposto é interativo, incremental guiado pelos objetivos a que se destina o sistema. Onde os modelos serão aperfeiçoados sucessivamente de acordo com que for aumentando o grau de abstração do sistema. O sistema será desenvolvido inicialmente de forma linear, só que sempre retornando ao nível anterior para modelar um conhecimento que foi esquecido. Ele poderá voltar múltiplas vezes para o ciclo inicial se assim for necessário. O sistema vai evoluindo à medida que cada ciclo é incrementado e completado. Isso faz com que só passe para o próximo nível se já tiver o domínio do conhecimento necessário, assim como evita que um nível do ciclo de desenvolvimento fique incompleto. Os modelos de cada nível lidarão apenas com a complexidade do conhecimento necessário para sua construção. Isso facilita o feedback trazendo mais informações e melhora o processo de análise nos ciclos subseqüentes. Os requisitos necessários ao sistema podem, aos poucos, irem sendo ajustados para se adequarem melhor às mudanças de acordo com a necessidades do projeto do sistema. O sistema é refinado e ajustado sempre que necessitar de novos requisitos. Essa opção por um ciclo de vida interativo e incremental evita as desvantagens de um ciclo de desenvolvimento em cascata, que é aquele que envolve um único passo de análise, de projeto e de construção. No ciclo de desenvolvimento em cascata tem como conseqüência uma sobrecarga na análise já que é necessário atacar toda complexidade do problema de uma vez. Tem uma realimentação retardada e corre o risco de perder todo trabalho se houver uma mudança no intervalo da construção do sistema.

Como processo do *framework* usamos as etapas do desenvolvimento de sistemas computacionais, regidos pelos princípios da metodologia KADS. Segundo a metodologia KADS os sistemas baseados em conhecimento deverão ser desenvolvidos considerando todas as etapas do seu ciclo de vida.

A principal contribuição do *framework*, é que o mesmo sirva de guia para construção de sistemas, já que a UML é apenas uma linguagem de modelagem, e não dá uma descrição lógica de que passos devem seguir para construção de um sistema. Ou seja, o vocabulário e as regras da linguagem de modelagem da UML indicam como criar e ler modelos bem formados, mas não apontam quais os modelos que devam ser usados para se ter à abstração necessária na construção de um sistema. Assim como também deixa em aberto em que instante da construção do sistema, esses diagramas devem ser construídos. Segundo [15] "A UML especifica uma linguagem para modelagem, mas não descreve explicitamente um procedimento de utilização indicando uma ordem de atividades que devem ser realizadas para se alcançar o objetivo final que é o de desenvolvimento de sistemas. Há, portanto a necessidade de se adotar um processo para utilização da UML". O *framework* tenta sanar esse problema, já que deixa claro no desenvolvimento do sistema onde se deve encaixar os modelos dentro do ciclo de desenvolvimento da metodologia KADS. Isso de acordo com a complexidade do sistema. Já na UML isso fica em aberto, pois, eles encaixam os modelos apenas no processo de análise de requisitos e análise de projeto. Ficando a cargo do desenvolvedor visualizar dentro do processo de desenvolvimento do software, as fases de desenvolvimento do sistema, já que sua abordagem é pouco expressiva em termos de processo. Isso é conseqüência de que nem sempre quem usa a UML usa os processos PMR (Processo de modelos comumente reconhecidos) [10].

Baseado nessa carência se procurou desenvolver o *framework* de forma a auxiliar tanto em termos de visualização do sistema, como dar ênfase à parte de processo especificado em que momento os diagramas deveriam ser construídos e em que eles contribuíam para o desenvolvimento do sistema. Assim como acrescentamos regras de produção ao *framework* para melhorar a base de dados. O *framework* proposto será mais uma ferramenta que pode ser utilizada na construção de sistemas. Já que ele tem uma notação híbrida, onde se procurou utilizar as melhores contribuições de vários métodos e metodologias existentes e consolidadas. Para a construção de sistemas que sejam mais adequadas a tecnologia OO, ainda não existe uma metodologia que o desenvolvedor possa seguir rigorosamente para construção de sistemas. A UML é uma linguagem gráfica que permite comunicar certos conceitos mais claramente do que as linguagens alternativas, que são normalmente usadas na construção de sistemas.

uso podem contribuir tanto nos aspectos estruturais do sistema como nos aspectos comportamentais. Em conjunto essas visões representam a base do projeto de software de qualquer sistema computacional.

5.3. Justificativas

O framework aqui proposto reune diversas características de metodologias, de métodos e ferramentas que possam auxiliar na construção de sistemas. Como as metodologias de desenvolvimento de sistemas consideradas não trabalham adequadamente a fase de elicitação do conhecimento e assumem que "o que" precisa ser representado a priori já é conhecido, existe uma grande dificuldade na hora de codificar a informação em uma estrutura de dados, ou seja na transformação e abstração do conhecimento, porque o sistema não utiliza o conhecimento de uma forma inteligente.

Segundo o enfoque que é dado no *framework* aos casos de uso, o conhecimento já vem sendo refinado e extraído no nível epistemológico na análise de requisitos, para assim, no nível lógico, ocorrer a transformação do conhecimento. Onde se acrescentam regras de produção para relacionar dados e extrair novas informações, a partir das que já foram armazenadas no sistema. Ou seja, usando regras de produção incorpora-se ao sistema um conhecimento que vai nos permitir inferir dados não explicitamente colocados no sistema e em alguns casos o conhecimento impreciso dos especialistas [5].

A principal contribuição do *framework* é facilitar a construção de sistemas orientados a objetos, através de uma nova abordagem, já que no nosso *framework* conseguimos fechar o ciclo do gráfico de transformação do conhecimento da KADS, com um novo enfoque iniciando as pesquisas de uma nova metodologia, com uma abordagem híbrida baseada no paradigma conceitual apesar de tratar os conceitos como objetos. Isso irá facilitar a construção principalmente de grandes sistemas, já que os desenvolvedores vão necessitar de um menor esforço mental, para mapear e simular o sistema a ser concebido. Assim como irá facilitar a visão do sistema como um todo, já que todos esses conceitos já foram mapeados conceitualmente de forma a evitar inconsistências, incertezas e erros no conhecimento adquirido. Esse conhecimento adquirido trará um melhor modelo de solução para o sistema.

Dentro dessa nova visão que é dada, quando se direciona a modelagem conceitual qualitativa, isso irá ajudar a construir apenas os modelos necessários à construção do sistema, assim como ajuda a definir a hora de encerrar a modelagem conceitual. Só voltando a modelar se faltar conhecimento na análise de requisitos. Os diagramas de caso de uso serão mais fáceis de serem construídos, já que serão feitos a partir de um modelo conceitual e não de um modelo mental. Assim como irá ajudar toda o processo de análise e de construção dos outros diagramas da UML, reduzindo assim o tempo gasto na construção de um grande sistema com um grande ganho na análise de custo/beneficio.

5.3.1. Justificativa do Uso do Gráfico de Transformação do Conhecimento da KADS

Por que não utilizar todos os modelos da KADS e sim apenas o ciclo de vida para desenvolvimento do *framework*? Ao definirmos o ciclo de vida podemos dizer que ele é o passo mais importante na definição de um software. Já que a construção de sistemas normalmente é feita com base em um modelo de ciclo de vida, onde deve ser adequado á situação do paradigma que se quer representar. O interesse pelo o ciclo de vida da KADS foi resultante da fácil visualização da transformação do conhecimento em seus níveis de abstrações, mas se não foram usados todos os modelos da KADS, foi porque foram detectadas algumas falhas neles tais como: No âmbito geral, a metodologia KADS tem uma boa descrição do modelo de problema, do modelo de solução e uma boa identificação das relações entre conceitos. Mas não tem totalmente uma boa interação entre os modelos e sua descrição do processo de abstração é parcialmente satisfatória.

Os modelos da KADS trabalham apenas parcialmente o nível lingüístico e conceitual. Conseqüentemente ela não trabalha completamente todos os passos na construção de um sistema. Em nenhum momento sua metodologia cita, como se pode melhorar a percepção no nível lingüístico. Eles erram ao pressupor que quando vai se construir um sistema, já se tem um mapeamento inicial do conhecimento do sistema a ser construído.

A KADS tem seu enfoque no conceito, mas não trata os conceitos como objetos dentro do paradigma orientado a objeto, fugindo assim do objetivo do *framework* já que o mesmo está direcionado para análise orientada a objeto. Para se construir um sistema existem três visões, a estrutural, a dinâmica e a visão funcional. Assim, a KADS não pode dar uma visão unificada nos diferentes aspectos envolvidos no sistema.

bancos de dados orientados a objetos que permita uma interação com a linguagem lógica usada para escrever as regras de produção (fuzzy ou não), um bom ambiente de programação para isto é o LPA Prolog (http://www.lpa.co.uk), pois suporta lógica fuzzy (módulo Flint), orientação a objetos (módulo Prolog++) e têm interface com diversos bancos de dados (*Microsoft Access, Btrieve, dBase, Ingres, Netware SQL, Oracle, Paradox, Progress, SQLBase, SQL Server, Sybase* entre outros), etc [18]. Após a implementação, se faz necessária que sejam feitos no sistema, manutenção preventiva para manter o sistema funcionando a um nível satisfatório. Isso para evitar que, ocasionalmente defeitos no programa que passaram despercebidos pelo teste do sistema sem terem sido aferidos. A correção de tais erros é uma função da manutenção. O sistema bem projetado terá de prever e dar margens às eventuais mudanças que possa sofrer o sistema.

5.2. Análise e Projeto da Construção de Sistemas no *Framework*

A análise e projeto são os pontos reais mais questionáveis no desenvolvimento de software, pois é, necessário se construir um código executável para poder se construir um software que realmente seja executável.

A linguagem de modelagem da UML, embora não muita bem definida em termos de processo, orienta iniciar seus modelos de uma maneira simples, tornando mais complexo de acordo com que os modelos vão evoluindo. Ela se baseia no paradigma conceitual apesar de tratar os conceitos como objetos. Os diagramas da UML permitem comunicar certos conceitos mais claramente do que algumas das linguagens de modelagens de sistemas alternativos, normalmente usadas na construção de sistemas e mesmo que lidando muito pouco com a linguagem natural, tem uma boa visão lógica. Apesar de ser um pouco imprecisa, quando trata de conceitos mais complexos, a UML com seus modelos ajuda a obter uma visão geral do sistema, mas seu modelo conceitual não tem um alto grau de abstração do conhecimento do mundo real. Da forma que seus modelos são feito, existe apenas o conhecimento necessário na construção de um sistema, onde nos mesmos já se tem embutido um pouco de abstração lógica. Quando os modelos conceituais são descritos em linguagem natural, eles conseguem ter um alto grau de abstração do mundo real, facilitando assim o entendimento do problema antes de obter a abstração necessária para construir o sistema.

Segundo [10] "Na UML, um modelo conceitual é exibido como um conjunto de diagramas de estrutura estática, nas quais não se definem operações. O termo modelo conceitual tem a vantagem de enfatizar fortemente os conceitos do domínio e não entidades de software". Em toda sua documentação sente-se a preocupação dos autores da UML em trabalhar o nível conceitual, apesar de ser considerada superficial a forma como os mesmos trabalharam.

Quando se analisa um diagrama de classe, ele dá a visão estática do sistema, e diz que tipos de abstrações estão presentes no sistema. Já quando se analisa um diagrama de caso de uso, tem-se uma visão instantânea de um aspecto do sistema que está sendo construído. Os diagramas de caso de uso são importantes para os requisitos do sistema. Eles dirigem todo o processo de desenvolvimento do sistema, fornecendo uma interação típica que o usuário tem com o sistema a fim de atingir um objetivo. Os casos de uso são elementos chave do sistema, pois facilitam que o usuário possa entender o mesmo, e ao mesmo tempo define que função tem importância para o usuário. Eles são importantes porque fornecem a base de comunicação entre clientes e desenvolvedores no planejamento do projeto.

Um caso de uso contem um conjunto de diagramas amarrados por um objetivo comum do usuário. Esses conjuntos de diagramas fornecem cenários que são uma seqüência de passos que descrevem interações entre o usuário e o sistema. O cenário é uma alternativa do que pode acontecer naquele processo. No entanto alguma coisa pode falhar e isso já traria outro tipo de cenário. Quando se juntam todos os diagramas de caso de uso pode-se obter a imagem externa do sistema e saber se o que o usuário realmente quer, está sendo projetado. Todo esse processo é lento já que é necessário descrever todo o processo do sistema. Uma das coisas mais importantes que deve ser feita na fase de elaboração do sistema é descobrir quais os casos de usos que são cruciais para o sistema que se está construindo, particularmente devem ser feitos os casos de uso mais relevantes e os mais arriscados. Pode-se fazer casos de usos através de um formato simples, onde se descreva o seu cenário primário, como uma seqüência de passos enumerados e as seqüências alternativas como variações naquele cenário.

Para compreender a arquitetura dos sistemas de softwares OO, se precisa recorrer a várias visões que são complementares e inter-relacionadas, assim como a visão dos diagrama de caso de uso, que expõe os requisitos do sistema. Ele fornece a visão do projeto captando o espaço de problema e espaço de solução. Outra visão dos casos de uso é na implementação do sistema, onde ela tem seu foco voltado para questões de engenharia de sistemas. Os casos de

- No **nível epistemológico** serão construídos os diagramas de caso de uso, com um enfoque mais específico de forma a dar a visão particionada necessária à construção do sistema. Eles serão desenvolvidos como o objetivo de captar o comportamento pretendido do sistema, isso sem nenhum detalhe de implementação. É nessa fase que se extrai dos conceitos utilizados nos diagramas de caso de uso específico o cenário para visualizar o que será necessário para implementar o sistema. Eles devem ser descritos, sempre considerando o que o sistema irá fazer tanto do ponto de vista externo como interno, dependendo do tipo de caso de uso do sistema. Eles conseguem visualizar o que está fora do sistema e como o sistema reage a algo externo, mas não é possível visualizar como o código pode ser executado. Os casos de uso envolvem a interação dos atores com o sistema. Um ator representa um conjunto coerente de papeis, que os usuários dos casos de uso desempenham quando interagem com esses casos para descrever um cenário. Os cenários descrevem o fluxo de eventos de um caso de uso. Os diagramas de caso de uso entram tanto na análise de requisitos como na análise da construção do sistema como um todo. Em seguida deveremos fazer a validação informal dos requisitos propostos, isto é averiguar se o modelo conceitual cobre todos os conhecimentos necessários para construir os casos de uso, assim como verificar se os casos de uso construídos dispõem dos requisitos necessários para construção dos modelos da próxima fase. Sempre que necessário deve-se retornar para o inicio do ciclo de vida para trabalhar um conceito que foi esquecido ou não foi aprofundado como deveria ter sido.

- No **nível lógico** usa-se a notação gráfica da UML, na construção de diagramas, para que assim, se possa descrever o cenário do sistema e obter a visão do que se espera que o sistema faça, isto é, os requisitos necessários para a construção do sistema. Podendo optar dentre os diagramas abaixo aqueles que podem melhor ajudar na visualização do sistema. Seus modelos devem ser inicialmente simples para irem ficando mais complexos de acordo com a complexidade do sistema. Depois de ter sido criado os diagramas de casos de uso no nível epistemológico, serão desenvolvidos os diagramas de interação de estado e de atividades. Os diagramas de interação descrevem a interação entre os vários objetos de um sistema. A principal utilidade desse diagrama é captar a seqüência de mensagens enviadas entre os objetos e mostrar o funcionamento dos casos de uso em termos de suas interações entre as classes. Quem vai ditar quantos diagramas serão necessários se construir é a complexidade do sistema, avaliando sempre custo/ beneficio, já que quanto melhor a visão do sistema mais fácil será implementa-lo, mas isso pode aumentar os custos do projeto, assim como aumentar o tempo que será necessário ser gasto na construção do sistema. Na representação do conhecimento é usado um ambiente híbrido com diagramas de classe e regras de produção. A necessidade de usar esta representação híbrida é, porque o conhecimento nos diagramas de classe da UML, às vezes não absorve todo o conhecimento da modelagem conceitual, depois que se criam as classes do diagrama de classe. O diagrama de classe é importante porque diminui a complexidade das informações e representa o conhecimento de forma sucinta.

    - Os diagramas de classe descrevem o conhecimento envolvido no escopo do problema, e assim demonstrar a estrutura estática do sistema. Estes diagramas apresentam as várias classes, atributos e operações do sistema assim como faz o mapeamento dos relacionamentos entre as classes.
    - As regras de produção fazem o embasamento lógico da parte do sistema que estava sendo perdido nos diagramas de classe. As regras fazem inferências de fatos novos a partir de fatos que fazem parte da base de conhecimento, ou seja, não é necessário colocar explicitamente todos os dados, mas só alguns deles e regras que possibilite deduzir o resto. Isto diminui o problema de integridade ao se adicionar novos dados e também o excesso de informações que normalmente geram redundâncias no sistema. Em certos tipos de problemas é aconselhável usar regras de inferências fuzzy [11].

- No **nível implementacional,** são usados modelos para ajudar na codificação do que foi mapeado em outras fases da construção do sistema. É um nível mais próximo do nível interpretado pela máquina, onde a implementação do modelo deve ser orientada a objeto. A fase implementacional visa solucionar o que o sistema se propõe e assim gerar os dados e a descrição da interface do programa, (desenvolvidos em módulos particionados), assim como o projeto de estrutura dos dados desenvolvidos anteriormente e as normas de arquiteturas aplicáveis. Logo depois têm-se os códigos e testes onde, serão construídos os programas de acordo com o projeto do programa. Cada módulo deverá ser codificado e testado em separado, para aos poucos serem integrado. Neste passo a codificação só estará completa quando o programa for testado em separado. A integração dos programas consiste em juntar os vários módulos de programa com o objetivo de construir um sistema. O primeiro passo dessa integração se inicia com os programas que já foram testados, cada um por si, e com dados que já foram verificados. A integração dos módulos e o passo que verifica se os programas que foram particionados estão corretos e se trabalham em conjunto para construir o sistema requerido pelo patrocinador. Resumindo essa integração é o teste para saber se o sistema cumpre o objetivo proposto. A opção para implementar o sistema deve ser uma linguagem de programação ou

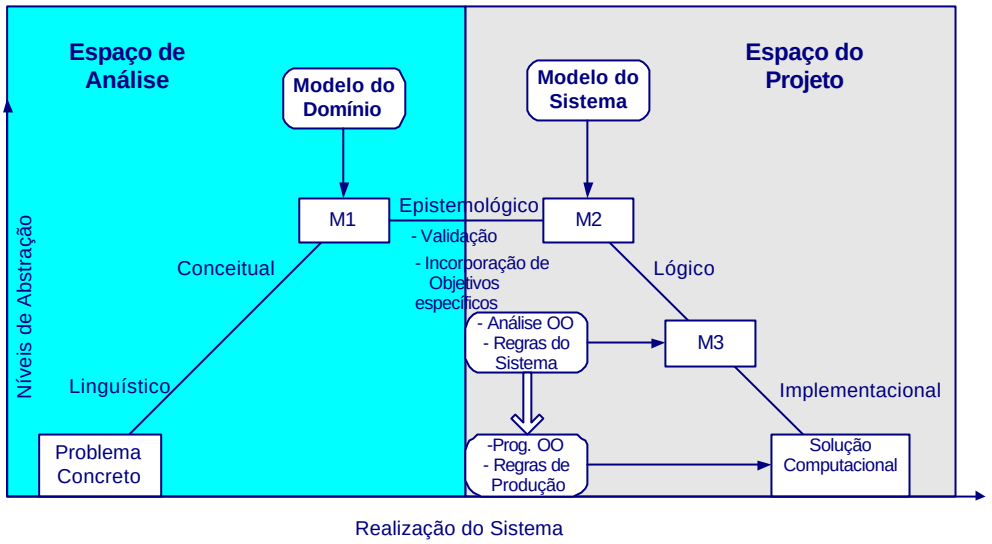

Figura 03 – Ciclo de Vida do *Framework*

Ao desenvolver um sistema complexo às vezes se faz necessário uma visão panorâmica do sistema, com o intuito de ajudar os investidores a visualizar a aparência e o funcionamento do futuro sistema e assim se chegar a um consenso nas negociações. Em qualquer situação os melhores tipos de modelos serão aqueles que permitem a escolha do grau de detalhamento, dependendo para quem seja a visualização e qual o objetivo. O patrocinador do sistema moverá o foco para a maneira como esses objetos funcionarão, enquanto, o desenvolvedor dirigirá sua atenção para o que ele quer visualizar. Isso nos leva a concluir que dependendo de quem observa o sistema pode ou não, obter uma visão distinta em diferentes tipos de abstrações.

5.1. Passos do *Framework* Proposto

Descreveremos o sistema de acordo com as diretrizes do ciclo de vida da KADS, para assim ir abstraindo o conhecimento em todos os seus níveis, conforme os passos a seguir.

- O **nível lingüístico** está dentro do espaço de análise, e é nele que se terá que abstrair todo o conhecimento referente ao problema real que se quer resolver. Buscando conhecer todo o ambiente relacionado ao domínio de interesse, para que assim, se tenha uma visão do escopo do problema em questão. Deve ser feito um estudo definindo a técnica de aquisição do conhecimento adequada de acordo com os objetivos definidos para assim iniciar a coleta de dados. Precisa-se ter parâmetros que auxiliem na representação do modelo mental, para auxiliar o Engenheiro do conhecimento a mapear o problema em questão, usando técnicas de elicitação do conhecimento como, por exemplo, entrevistas, observação, análise de protocolos, questionários, discussão com o perito, elicitação construtiva, etc [6,7].

- No **nível conceitual** usa-se a modelagem conceitual qualitativa, na qual não são usados formalismos para seu desenvolvimento, e sim, uma semântica livre para lidar melhor com a diversidade das informações. Na aquisição do conhecimento é feita a verificação de quais os conceitos e atributos que estão relacionados com o escopo em questão, para assim fazer uma conexão entre essas informações. Os conceitos mais genéricos são colocados na parte de cima do modelo, pois eles ajudam a definir os atributos quando for passar para a fase de representação do conhecimento. Quando são definidos os detalhamentos dos conceitos, procura-se sempre levar em consideração que quanto mais informações se tiverem do escopo do problema, mais fácil será alimentar e desenvolver o sistema que está sendo construído. Na modelagem conceitual procura-se ser flexível, para assim permitir a adição, remoção e modificação do conhecimento mapeado.

ainda não tem uma maturidade na construção de sistemas. A UML utiliza o RUP (*Rational Unified process*) onde o processo é descrito detalhando sobre que tipos de modelos devem ser usados nos vários estágios do processo. Entretanto, eles colocam que não é obrigatório utilizar o RUP já que eles são distintamente separados. Isso traz como conseqüência no desenvolvimento de sistemas práticos a necessidade de um processo que englobe o melhor de vários métodos e metodologias existentes para se trabalhar com o paradigma orientado a objeto

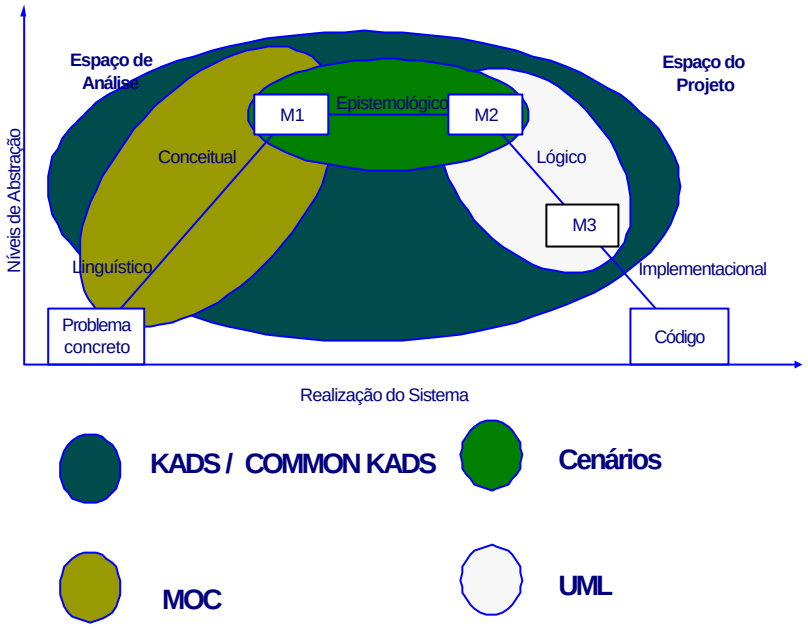

Figura 02 – Gráfico de Transformação do Conhecimento de diversas metodologias

O RUP é um processo, ou seja, um conjunto de passos parcialmente ordenados com a intenção de atingir a meta de entregar, de maneira eficiente e previsível, um produto de software capaz de atender as necessidades de seu negocio. O RUP consiste em uma abordagem de um ciclo de vida, o qual os autores da UML consideram que seja adequado a ser utilizado junto com a UML. Ele captura algumas das melhores práticas atuais de desenvolvimento de software e tenta abranger uma variedade de projetos de sistemas. Ele é centrado na arquitetura, ou seja, o processo focaliza o desenvolvimento inicial e a linha de base da arquitetura de um software. A principal vantagem disso é dispor de uma arquitetura robusta capaz de facilitar o desenvolvimento paralelo, assim como minimiza a necessidade de refazer o trabalho no caso de uma alteração. Ele aumenta a probabilidade de reutilização de componentes e a capacidade de manutenção eventual de um sistema. As atividades de desenvolvimento do RUP são orientadas a casos de uso. Ele atribui uma forte ênfase à elaboração de sistemas com base em uma compreensão completa a respeito de para que o sistema se destina a fazer e como será utilizado. Esse processo dispõe de suporte de técnicas orientadas a objetos. Os modelos se baseiam nos conceitos de objetos e classes e nos relacionamentos existentes entre eles. O RUP utiliza a linguagem de modelagem UML como sua notação.

5. O FRAMEWORK PROPOSTO

O *framework* proposto para desenvolvimento de sistemas complexos, visa guiar o processo de solução destes problemas, procurando tratar o conhecimento de forma a facilitar a evolução das abstrações em todos os níveis do ciclo de vida de um sistema. O *framework* segue o ciclo de vida da metodologia KADS, porém indicando que modelos devem ser usados a cada fase do ciclo.

Considerando os parâmetros usados para comparação do estudo de metodologias citadas anteriormente, se chegou à seguinte conclusão: nenhuma das quatro metodologias vistas atingem eficientemente os quatro itens comparados, havendo portanto necessidade de desenvolver um método que seja eficiente na construção de sistemas. Segundo esse estudo comparativo entre metodologias, Souza afirma "que as metodologias estudadas, apesar de atenderem corretamente aos seus paradigmas, não nos pareceram adequadas para tratar os requisitos, por nós identificados, como eficientes para lidar com o desenvolvimento dos sistemas dinâmicos" [17]. No estudo comparativo entre metodologias, foram considerados os quatro itens abaixo, comparando como as metodologias estudadas focalizam as perspectivas estruturais que tratam do problema; dinâmica que trata da solução e funcional que trata da abordagem sistêmica e genérica das duas perspectivas anteriores. O estudo também considerou a possibilidade de integração entre as perspectivas citadas anteriormente e a possibilidade de usa-las para modelar sistemas dinâmicos.

Concluiu-se também que as quatro metodologias, por terem abordagens diferentes, tratam o problema de forma diferente. De acordo com o resultado dessa pesquisa as metodologias MOC e Cenários não se preocupam em conduzir a elaboração e compreensão de um modelo mental, ela pressupõe em sua definição, que o modelo conceitual já deveria estar pronto quando se opta por utilizar o paradigma adotado por elas. A KADS e Common KADS, ainda que de uma maneira não planejada, conseguem mapear superficialmente este modelo, quando se propõe descrever os conceitos e atributos que irão servir para o modelo conceitual do conhecimento. No nível lógico as metodologias KADS e Common KADS conseguem ser mais eficientes, pois possuem passos que sendo seguidos se chega à representação do conhecimento.

No nível implementacional as metodologias não formais não conseguem orientar de forma concreta, elas apenas estabelecem parâmetros a serem utilizados para a implementação. A KADS e a Common KADS se aproximam mais, quando orientam a utilização do uso do modelo de sistema e do usuário onde sugere a codificação do conhecimento, apesar de não fazer isso de forma concreta, já que apenas estabelece parâmetros que podem ser utilizados para se chegar à implementação.

Ainda como conclusão desse estudo teve-se, que as metodologias KADS e Common KADS, por serem mais formais, conseguem se adequar melhor à construção dos diferentes modelos gerados em cada fase do ciclo de vida. Apesar de muitas vezes seus modelos se comportarem de forma superficial, sem clareza e com pouca generalização. A maioria das metodologias são direcionadas à área de computação assumem que a priori, já exista modelos mentais e conceituais do que se deseja construir, e abordam muito superficialmente esses aspectos no ciclo de vida, conforme estudo comparativo entre metodologias [17]. Ela também afirma que se precisa desenvolver uma "extensão futura, pensando em utilizar os resultados obtidos para propor pesquisa para a criação de uma metodologia de modelagem que possua um paradigma mais qualitativo, e que, a partir da fusão das abordagens aqui estudadas, consiga ser mais genérica e abrangente para ser usada na construção dos sistemas dinâmicos e complexos, tratando especialmente da dinamicidade do problema e da concepção/geração de soluções igualmente dinâmicas".A figura 02 apesar de aparentemente demonstrar que a KADS e a Common KADS, conseguem abranger todos os níveis do gráfico da transformação do conhecimento, concretamente ela se adequam melhor na construção dos diferentes modelos gerados em cada fase. Mas isso não quer dizer que elas conseguem abranger satisfatoriamente o conhecimento, pois ao contrario do que possa parecer, esses modelos muitas vezes se comportam de forma superficial, sem clareza ou de forma muito especifica, com pouca generalização conforme concluído anteriormente. Já as outras duas metodologias, a MOC e a cenários (na figura 02) só foram demarcadas, na parte que elas se concentram de forma mais eficiente, já que tratam nesses níveis o conhecimento com mais profundidade.A priori a UML necessita que o desenvolvedor já tenha uma estrutura cognitiva bem elaborada sobre o sistema a ser desenvolvido. Ou seja, apesar dela se basear no paradigma conceitual, mesmo tratando os conceitos como objeto, ela orienta o desenvolvedor a fazer um modelo conceitual do sistema que está sendo construído, mas de uma forma muito superficial explorando muito pouco o conhecimento. Isso mostra que a visão da UML apesar de todos o seus modelos de análise é direcionada à implementação. Segundo [10] "UML é uma linguagem para modelagem; ela não guia um desenvolvedor em como fazer análise e projeto orientados a objeto, ou qual o processo de desenvolvimento a ser seguido". Segundo [15] "A modelagem em UML a partir do mundo real é, provavelmente mais difícil, pois, precisa de um maior esforço mental por parte dos projetistas, a fim de mapear e simular na mente o sistema desenvolvido". Ela ainda afirma que "para um sistema relativamente complexo esta hipótese é inviável, já que a grande quantidade de conceitos e associações entre conceitos fazem com que se perca a visão do sistema como um todo, ou mesmo, de parte do sistema".

A parte de processo é fundamental em uma metodologia para construção de sistemas, e a UML não trabalhou bem esse aspecto, deixando muitas dúvidas em alguns desenvolvedores e principalmente na área acadêmica que o aluno

descrição da implementação do sistema, independente do domínio do conhecimento em um nível epistemológico. A KADS pressupõe que as construções de sistemas obedecem necessariamente à criação de modelos que são desenvolvidos em etapas onde a cada passo são identificados os níveis de abstração do conhecimento.

A KADS propõe um ciclo de vida, prevendo um modelo baseado em etapas nas quais a transformação do conhecimento vai acontecendo de acordo com a abstração do conhecimento na evolução de um sistema inteligente. Baseada no seu ciclo de vida, ela disponibiliza técnicas de construção de modelos que auxiliam na construção dos sistemas, transformando o conhecimento de forma a obter o menor número de perdas possíveis.

3. MODELAGEM CONCEITUAL QUALITATIVA

A MCQ (modelagem conceitual qualitativa) [5,15] ou também denominada modelagem conceitual cognitiva [16] é uma técnica que tem um grande poder de expressividade, já que o formalismo utilizado é simples e sua notação gráfica é flexível. Ela permite o uso de termos claros, o que facilita o entendimento do problema quase tão bem quanto a linguagem natural. A MCQ incorpora os requisitos não funcionais do sistema, permitindo trabalhar na análise de domínio ou na análise de requisitos. Esse processo de modelagem trabalha a partir do conhecimento, tentando conservar as informações mais relevantes do domínio em escopo em questão.

A MCQ não só deve ser usada como técnica para trabalhar o modelo conceitual na construção de sistemas, como pode ser usada para reorganizar uma empresa ou para ajudar na tomada de decisões de uma organização.

Segundo a notação de [15] dá a modelagem conceitual qualitativa usa as seguintes convenções:

- **Conceitos**: são representados por elipses contendo em seu interior o nome do conceito.
- **Relacionamentos**: são representados por linhas direcionadas e rotuladas entre os conceitos. Onde o rótulo pode indicar hierarquia ou influência entre os conceitos.
- **Atributos**: são conceitos, porém não precisam ser expandidos, pois, são conceitos de complexidade mínima no escopo em questão. São representados por nomes ligados aos conceitos através de relacionamentos sem rótulos.

4. UMA ANÁLISE SOBRE METODOLOGIAS DE MODELAGENS DE SISTEMAS

Ao longo da história e com o avanço da tecnologia, várias metodologias foram criadas, em diversas áreas de acordo com suas necessidades. Elas adquiriram três formas diferentes, já que foram influenciadas por paradigmas de programação diferentes. As metodologias estruturadas seguem o paradigma das linguagens procedurais, enquanto a linguagem de modelagem UML segue o paradigma das linguagens orientadas a objetos. Já as metodologias orientadas a dados foram influenciadas tanto pelo o paradigma de dados convencionais como pelos orientados a objetos. Essas variedades de abordagens das metodologias dão um enfoque diferente para cada uma delas. Isso dificulta o desenvolvedor, na escolha de uma metodologia quando vai construir um sistema. Quando se tem conhecimento do seu enfoque, se pode distinguir que modelos são positivos e quais modelos são negativos e fazer a escolha do método mais adequado ao seu caso, ou trabalhar com uma metodologia híbrida considerando sempre o que melhor se adeqüa ao seu caso. Na área de informática várias delas foram desenvolvidas, sempre se propondo a reduzir a complexidade da construção de sistemas. Essas metodologias apresentam sempre abordagens diferentes e normalmente são influenciadas por paradigmas de várias linguagens de programação. Algumas metodologias apresentam modelos que se propõem a abranger todo o ciclo de vida da construção de sistemas. Outras têm apenas uma abordagem parcial, abordando alguns aspectos específicos do ciclo de vida.

Existem diversas representações de modelos para construção de sistemas computacionais, com diferentes enfoques de acordo com o seu ponto de vista. Nessas metodologias deve-se analisar dois aspectos: O processo que são os passos na construção de um sistema, e os tipos de conceitos que são usados na construção de um sistema. Esses modelos diferem por abordarem uma visão diferente quando classifica os diferentes níveis de abstrações necessários na construção de um sistema. Elas apresentam diferentes notações e conceitos, já que isso é determinado pelo paradigma das linguagens que elas representam. Enquanto algumas se apóiam em linguagens livres e pouco estruturadas, outras se apóiam em linguagens precisas e formais. As metodologias são baseadas em duas diferentes formas, sendo elas voltadas para o individuo e para o agente. A KADS é voltada para o individuo, já a Common KADS é voltada para a perspectiva de agentes.

1. INTRODUÇÃO

Na medida que cresce a preocupação com a necessidade de softwares, cada vez mais as organizações de desenvolvimento de software têm procurado seguir metodologias que ajudem na construção de sistemas. Foi assim que surgiram as metodologias direcionadas à construção de sistemas tais como: KADS (*Knowledge Acquisition and Design Structure*) [2,3,8], Common KADS [14] , Cenários, MOC (Modelo orientado por conceito) e UML (*Unified Modeling Language*) [1,4,10], entre outras. No entanto algumas dessas metodologias não deixam claro um roteiro que diga quais os passos a seguir para desenvolver um sistema, desde a concepção do problema até a sua solução computacional, pois geralmente as mesmas deixam lacunas ou começam assumindo que, se tem claro qual o problema a ser resolvido. Essas metodologias apesar de se proporem a serem utilizadas tanto na linguagem de modelagem, quanto no processo para construção de sistemas, normalmente estão direcionadas para diferentes enfoques de acordo com o seu ponto de vista. Elas abordam um aspecto específico de acordo com suas notações e conceitos, que são ditados pelo paradigma que elas representam. Enquanto outras metodologias têm um modelo conceitual, o qual detalha num nível muito baixo, com pouca abstração e lida com os requisitos de uma forma concreta. Assim se faz necessário uma metodologia que trate do problema desde sua concepção até sua implementação deixando completamente claro o roteiro a seguir, trabalhando com um alto grau de abstração nos requisitos para um melhor entendimento do problema. De fato, o próprio grupo que desenvolve a Common KADS há 15 anos, tem sentido esta necessidade e tem incorporado à Common KADS notações compatíveis com UML, tais como diagramas de classes, diagramas de atividade e diagramas de estados (para um maior aprofundamento veja o livro [14] ou o site http://www.commonkads.uva.nl/page-commonkads.htm). No entanto casos concretos que usem diagramas da UML em sistemas baseados em conhecimento usando a metodologia Common KADS, é ainda um campo de pesquisa.

Neste trabalho, propomos um *framework* baseado em métodos e metodologias já existentes, que em sua maioria já vem sendo utilizadas em separado, pela comunidade que desenvolve sistemas computacionais. Esta abordagem híbrida visa, facilitar a construção destes sistemas, assim como tenta trabalhar melhor alguns níveis, que geralmente nas metodologias existentes tinham apenas uma abordagem parcial. Com essa nova abordagem se espera que alguns aspectos não muito claros para o desenvolvimento de sistemas possam ser melhor esclarecidos. O *framework* proposto será baseado no ciclo de vida da metodologia KADS, no sentido que ele será usado como um roteiro a ser seguido na construção deste tipo de sistemas. Além disso, para modelar os diversos níveis deste ciclo de vida, usará a linguagem de modelagem unificada UML, a modelagem conceitual qualitativa [6,15,16], e regras de produção que podem ser baseadas em lógica clássica ou fuzzy em caso de usar o raciocínio aproximado [11,18].

2. METODOLOGIA KADS

A KADS é uma metodologia que tem como objetivo ajudar na construção de sistemas baseados em conhecimento. Ela tem como meta principal separar a análise do projeto. Baseada nessa característica ela procura teoricamente descrever todos os passos que existem na construção de um sistema.

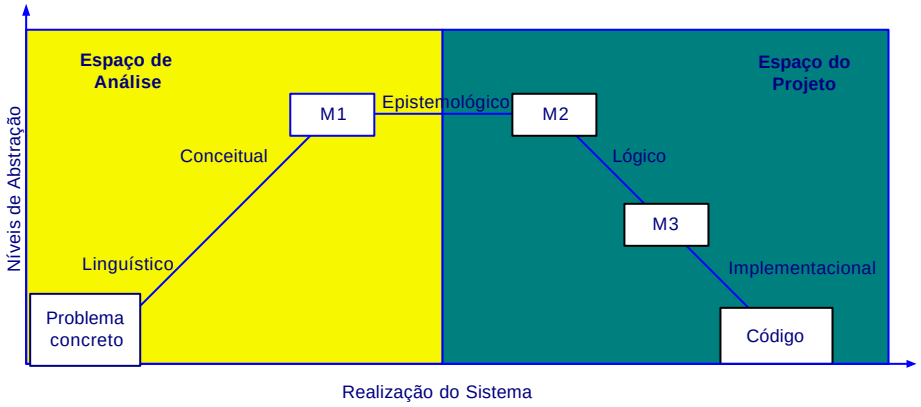

Figura 01 - Ciclo de vida da KADS

Esses passos servem de guia para o engenheiro do conhecimento na modelagem de dados verbais, que seria um passo intermediário na construção do sistema gerado por um modelo de interpretação. O modelo de interpretação é a

Um Framework para Desenvolvimento de Sistemas Complexos

Ivanosca Andrade da Silva
Ivanosca@dimap.ufrn.br

Benjamín René Callejas Bedregal
bedregal@dimap.ufrn.br

Márcia Jacyntha Nunes Rodrigues Lucena
marciaj@dimap.ufrn.br

Márcia de Paiva Bastos Gottgtroy
marcia@dimap.ufrn.br

Laboratório de Lógica e Inteligência Computacional – LabLIC
Departamento de Informática e Matemática Aplicada – DIMAp
Universidade Federal do Rio Grande do Norte – UFRN

RESUMO

Este trabalho, propõe um *framework* que proporcione diretrizes a serem seguidas para desenvolver sistemas complexos que requeiram uma grande quantidade de dados e o conhecimento de múltiplos especialistas. Neste sentido, o framework concilia aspectos da metodologia KADS, usada no desenvolvimento de sistemas inteligentes, com a linguagem de modelagem unificada UML, com regras de produção (baseados na lógica clássica ou *fuzzy*) e com a modelagem conceitual qualitativa.

Palavras chaves: framework, KADS, UML, desenvolvimento de sistemas complexos, modelagem conceitual qualitativa.

ABSTRACT

This work, considers one framework that it provides lines of direction to be followed to develop complex systems that require a great amount of data and the knowledge of multiple specialists. In this direction, framework conciliates aspects of methodology KADS, used in the development of intelligent systems, with the language of unified modeling UML, with rules of production (based on classic or fuzzy logic) and with the qualitative conceptual modeling.

Keyword: framework, KADS, UML, Development of complex systems, Qualitative conceptual modeling.