2
Revisão da Literatura

Um projeto normalmente está suscetível a sofrer alterações sobre seu planejamento prévio de atividades e recursos, o que pode ser denominado como reconfiguração dinâmica. A ferramenta proposta neste trabalho consiste em uma linha de produto de software de sistemas de gerenciamento de projetos de software com base em agentes inteligentes. Tal ferramenta visa apoiar o gerenciamento de projetos desenvolvidos segundo a metodologia de desenvolvimento Rational Unified Process, com foco na tentativa de resolução de alguns dos problemas gerados pela reconfiguração dinâmica dos mesmos.

Este capítulo apresenta um resumo teórico dos assuntos abordados no trabalho. Desta forma, a Seção 2.1 aborda a disciplina de Gerenciamento de Projetos. Na Seçõa 2.2 é apresentado o conceito de Reconfiguração Dinâmica. A Seção 2.3 traz a metodologia de desenvolvimento Rational Unified Process. A Seção 2.4 apresenta o conceito de Sistemas Multi-agentes. A Seção 2.5 discorre sobre Linhas de Produto de Software. Na seção 2.6 é apresentada a junção entre Sistemas Multi-agentes e Linhas de Produto de Software. Por fim, a Seção 2.7 traz algumas considerações finais sobre o capítulo.

2.1.
Gerenciamento de Projetos

O mundo de hoje é movido por projetos. Em diversas áreas de aplicação, produtos e serviços novos são produzidos através de projetos, o que pode ser demonstrado pelo aumento do número de empresas que estão adotando a metodologia de gerenciamento de projetos (Kerzner 2001). Segundo (ABNT 2000), o termo Projeto consiste em “processo único, consistindo de um grupo de atividades coordenadas e controladas com data para início e término, empreendido para alcance de um objetivo conforme requisitos específicos, incluindo limitações de tempo, custo e recursos”. O mesmo termo é definido pelo Project Management Institute (PMBOK 2004) como “um empreendimento temporário, planejado, executado e controlado, com objetivo de criar um produto ou serviço único”. De forma mais simplificada, podemos dizer que projetos

possuem duas fases distintas: a fase de planejamento e a fase de execução (Callegari e Bastos 2008). Ao longo da fase de planejamento as informações necessárias para o andamento do projeto são coletadas, como por exemplo as atividades a serem realizadas e os recursos necessários para execução das mesmas. Já a fase de execução envolve o gerenciamento, monitoração e controle do projeto em si até que o mesmo origine um produto final. Idealmente, a fase de execução deveria seguir os insumos da fase de planejamento, mas na prática esta não costuma ser a realidade da maioria dos projetos de software.

O gerenciamento de projetos consiste em coordenar atividades com o objetivo de atingir as expectativas dos envolvidos[1] (Patah e Carvalho 2002). Em outras palavras, é a arte de aplicar conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas para atingir um conjunto de objetivos pré-definidos. Tal disciplina tem o objetivo de manter os riscos de fracasso em um nível tão baixo quanto necessário durante o ciclo de vida de um projeto. O risco de fracasso aumenta com a presença de incertezas durante os estágios do projeto. O desafio de primeira linha do gerenciamento de projetos é atingir todos os objetivos do projeto enquanto preserva suas restrições (e.g. escopo, tempo e custo). Já o desafio de segunda linha refere-se à otimização da alocação e integração dos elementos envolvidos no projeto (e.g. recursos e atividades) necessários para atingir os objetivos pré-definidos (Phillips 2006).

Como disciplina, a gerência de projeto foi desenvolvida a partir de diversos campos de aplicação, como por exemplo o ramo da construção, da engenharia e de projetos militares, dentre outros (Cleland e Gareis 2006). Dois nomes importantes relacionados à disciplina são Henry Gantt, famoso pelo uso do gráfico de Gantt como ferramenta de gerenciamento de projetos, e Henri Fayol, criador das cinco funções de gestão que formam a base de conhecimento associada ao gerenciamento de projetos e programas[2].

No campo da informática, existem dois tipos de abordagens geralmente utilizadas no gerenciamento de projetos. As abordagens do tipo tradicional identificam uma sequência de passos a serem completados. Tais abordagens contrastam com a abordagem conhecida como desenvolvimento ágil de

[1] Pessoa ou entidade que afeta ou é afetada pelas atividades de um projeto.

[2] Grupos de projetos gerenciados de forma coordenada, visando obter benefícios difíceis de serem obtidos quando gerenciados isoladamente.

software, na qual um projeto é tratado como um conjunto de pequenas atividades e não como um processo completo. Tal abordagem é bastante controversa, especialmente quando se trata de projetos muito complexos. Dentre as abordagens do tipo tradicional, podemos citar a do PMBOK (PMBOK 2004), que tem se tornado um padrão de fato em diversas empresas e organizações. O gerenciamento dos projetos envolvidos neste trabalho é fundamentado na abordagem tradicional do PMBOK.

No tipo de abordagem tradicional, existem cinco grupos de processos no desenvolvimento de um projeto:

(i) Iniciação – são definidos os objetivos gerais, identificadas as principais premissas e restrições e estabelecidas as grandes limitações de orçamento e prazo.

(ii) Planejamento – envolve atividades como gerenciamento do escopo, do tempo, dos custos, dos riscos e da integração do projeto.

(iii) Execução – visa cumprir os objetivos e metas estabelecidas na fase de planejamento.

(iv) Monitoramento e Controle – acompanha a evolução do que é produzido no processo de execução através de indicadores que sinalizem a aderência das atividades executadas à base de referência para medição dos resultados e do desempenho do projeto.

(v) Encerramento – implica na aceitação formal do projeto e seu término.

Na Figura 2.1 os processos citados acima são exibidos. Nela é possível ver a relação existente entre esses estágios. Vale ressaltar que nem todos os projetos vão seguir todos estes estágios, já que projetos podem ser encerrados antes de sua conclusão.

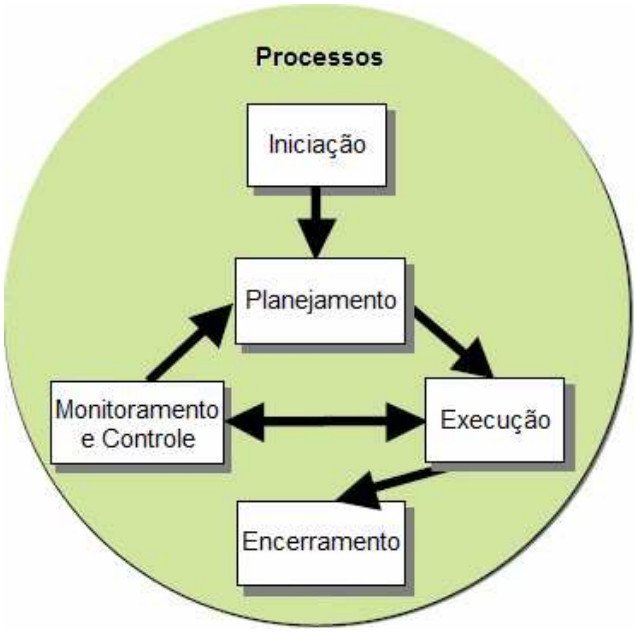

Figura 2.1: Processo de Desenvolvimento Tradicional.

Os projetos precisam ser executados e entregues seguindo determinadas variáveis. Tradicionalmente, o gerenciamento de projetos tenta manter o controle sobre três variáveis, conhecidas também como Triângulo da Gerência de Projetos, no qual cada lado representa uma variável. Um lado do triângulo não pode ser modificado sem impactar nos demais. As variáveis em questão são:

(i) Escopo – são as exigências especificadas para o resultado fim, ou seja, o que se pretende e o que não se pretende realizar.

(ii) Tempo – o tempo requerido para terminar os componentes do projeto.

(iii) Custo – o orçamento disponível para o projeto.

Um refinamento adicional das restrições separa a qualidade do produto do escopo, transformando esta em uma quarta variável. Contudo, a qualidade pode ser vista também como um dos principais componentes da variável de escopo. Na Figura 2.2 é possível ver a representação dessas variáveis.

Figura 2.2: Triângulo do Gerenciamento de Projetos.

2.2. Reconfiguração Dinâmica

Projetos de software normalmente estão bastante suscetíveis a passar por inúmeras modificações ao longo do seu ciclo de vida, uma vez que um projeto é em geral muito dinâmico e sua natureza particular demanda ajustes recorrentes em seu planejamento mesmo durante a fase de execução. Muitos deles falham ao atingir seus resultados necessários dentro do custo e do prazo planejados (Matiscola 2007). O desenvolvimento de projetos demanda um esforço singular que envolve lidar com atividades, recursos e fluxo de trabalho, dentre outros elementos, para se alcançar os resultados almejados. As empresas de software muitas vezes fazem uso dos conhecimentos de gerenciamento de projetos, bem como de processos de desenvolvimento, a fim de construir suas soluções com qualidade e dentro das restrições de tempo, recurso e escopo. O gerenciamento de projetos é uma questão fundamental para o sucesso do desenvolvimento de sistemas de software e tem como objetivo principal identificar e manter os riscos de fracasso em um nível tão baixo quanto necessário durante o ciclo de vida de um projeto. Exemplos de atividades essenciais e não triviais realizadas no gerenciamento de projetos são: (i) estimação de prazo e custos, a qual é diretamente afetada pelas características do projeto, quantidade de recursos alocados e pelo escalonamento de suas atividades, dentre outros; e (ii) replanejamento de atividades e realocação de recursos, dado que projetos de software tipicamente passam por diversas modificações ao longo do seu ciclo de vida, essas duas atividades tornam-se questões críticas. Além disso, quanto

maior o número de projetos envolvidos, maior a complexidade no gerenciamento dos mesmos (Schwalbe 2002).

Mesmo quando gerentes de projeto têm um bom entendimento acerca das dificuldades existentes em um projeto, eles frequentemente enfrentam problemas para estimar e justificar o custo e tempo para corrigir suas causas (Matiscola 2007). De acordo com (Joslin e Poole 2005), estimativas de duração de tarefas e recursos necessários na engenharia de software são muitas vezes imprecisas, enquanto o gerenciamento efetivo de um projeto de software deve ser muito dinâmico. Já em (Lee e Miller 2004) é dito que o gerenciamento de múltiplos projetos é crucial, visto que um projeto pode afetar a data de término de outros projetos e determinar o sucesso ou fracasso dos mesmos, devido à existência de recursos compartilhados. Além dos vários problemas envolvidos no gerenciamento de projetos, vale destacar que os mesmos possuem tamanhos diferentes. Projetos de pequeno porte não demandam tantas atividades quanto os de médio e grande porte.

Um projeto normalmente sofre perturbações externas, sejam elas voluntárias ou não. Em (Joslin e Poole 2005) é dito que abordagens puramente "estáticas" não resolvem os principais problemas no desenvolvimento de software. Com isso, as incertezas acerca de um projeto devem ser constantemente monitoradas e o planejamento ajustado durante todo o seu ciclo de vida (Joslin e Poole 2005). Os processos de desenvolvimento de software atuais auxiliam na resolução dessa questão por meio de refinamentos incrementais ou cíclicos do planejamento e definição do produto (Kruchten 1999) (Schwalbe 2002). O termo “reconfiguração dinâmica” no contexto pode ser definido por qualquer alteração feita sobre um planejamento prévio de atividades e recursos associados a um projeto. Em outras palavras, a reconfiguração dinâmica em projetos de software lida com eventos, ações, elementos afetados e consequências dos ajustes no processo de desenvolvimento de sistemas durante sua fase de execução (Callegari e Bastos 2008). Tal termo também tem sido utilizado em outras áreas, como computação autonômica (Horn 2001) e sistemas distribuídos (Coulouris, Dollimore e Kinberg 2002). Embora a definição atual possua pequenas diferenças das adotadas em tais áreas, o conceito principal do termo consiste na ocorrência parcial ou total de modificações durante a fase de execução. Atividades dos mais variados tipos devem ser associadas a recursos com características particulares a fim de atingir os objetivos relacionados a custo e prazo. O gerenciamento de projetos deve refletir dinamismo porque os projetos são intrinsecamente dinâmicos. Tal dinamismo

dificulta a adaptação do gerenciamento ao longo do desenvolvimento, uma vez que em geral a identificação e tratamento dos problemas acontecem tardiamente. Quanto mais tarde os problemas são sanados, mais difícil a recuperação do andamento normal do projeto e maior a chance de comprometimento no custo e prazo do mesmo. Devemos ainda considerar que decisões corretas individualmente podem não ter a mesma representatividade coletivamente.

Dessa forma, existem muitos problemas no gerenciamento em um cenário de multi-projetos que devem ser considerados e sua resolução é fundamental para garantir o sucesso do desenvolvimento do projeto:

(i) predição de prazos;

(ii) alocação adequada de recursos;

(iii) reconfiguração dinâmica, que envolve realocação de recursos e replanejamento de atividades;

(iv) uso de técnicas adequadas de acordo com o tamanho do projeto.

Apesar de cada um dos problemas citados serem individuais, é difícil tratá-los de forma isolada. Existe a necessidade de se buscar uma solução de forma coletiva, uma vez que cada um dos problemas pode afetar os outros (e.g. recursos compartilhados em atividades de projetos distintos). Muitos problemas envolvidos no gerenciamento de projetos vêm sendo largamente explorados na literatura. Soluções envolvendo heurísticas e outras metodologias são apontadas para solução dos mesmos (e.g. (Alcaraz e Maroto 2001)). Essas abordagens são complementares à solução aqui proposta, podendo ser integradas ao mecanismo de raciocínio dos agentes.

2.3. Rational Unified Process

O Rational Unified Process (RUP) é um processo de desenvolvimento de software que foi criado pela Rational Software Corporation, adquirida posteriormente pela IBM. É apoiado por diversas ferramentas de desenvolvimento integradas e vendidas pela IBM através de seus "Rational Suites". Tal processo fornece uma abordagem baseada em disciplinas para

atribuir tarefas e responsabilidades aos integrantes de uma organização de desenvolvimento (RUP 2001). O seu objetivo principal é assegurar a produção de software atendendo às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis. O RUP baseia-se no paradigma de Orientação a Objetos e a notação Unified Modeling Language (UML) (Booch, Rumbaugh e Jacobson 2000) para projetar e documentar os processos em ação.

Nas atividades do processo RUP, *modelos* são criados e mantidos (Gornik 2004). Ao invés de focar na produção de uma grande quantidade de documentos em papel, o processo sugere a concepção e manutenção de representações de modelos semanticamente enriquecidos do software em desenvolvimento. As diversas ferramentas providas pela IBM são utilizadas para lidar com os diversos artefatos – modelos em particular – do processo de engenharia de software (e.g. modelagem visual, programação e teste). Através de tais ferramentas, é possível controlar as mudanças sofridas ao longo do ciclo de vida do projeto, rastrear e atualizar os artefatos afetados por tais mudanças, além de apoiar o gerenciamento de configuração que acompanha cada iteração que compõem as fases do projeto.

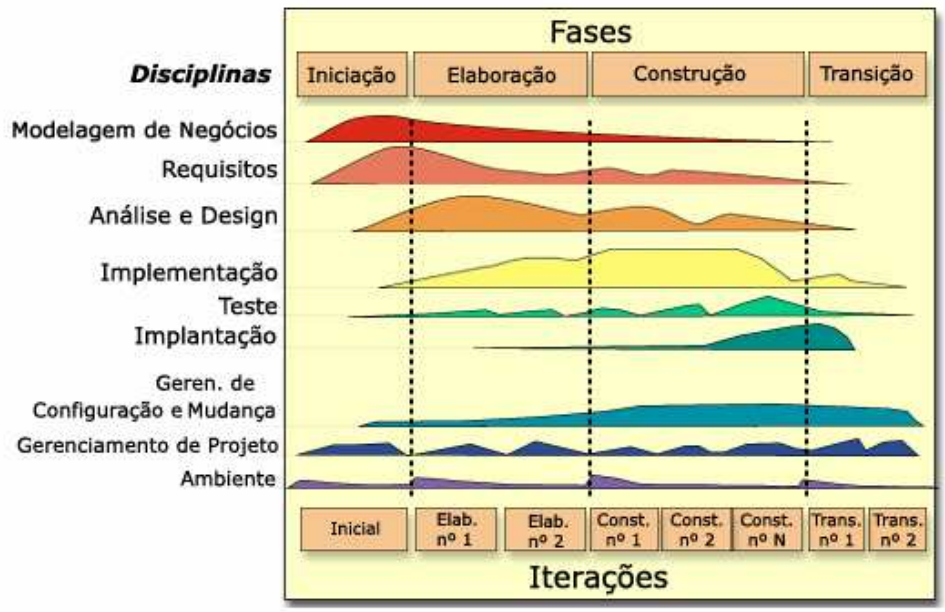

Figura 2.3: Gráfico das Baleias.

A Figura 2.3 mostra a arquitetura geral do processo, que possui duas dimensões. O eixo horizontal representa o tempo e mostra os aspectos do ciclo de vida do processo ao longo do seu desenvolvimento. Tal dimensão é expressa em termos de fases, iterações e marcos e representa o aspecto dinâmico do

processo. Já o eixo vertical representa as disciplinas, que agrupam as atividades por sua natureza. Tal dimensão é expressa em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis e representa o aspecto estático do processo.

Como dito anteriormente, o eixo horizontal do gráfico expresso na Figura 2.3 representa o aspecto dinâmico do processo ao longo do tempo. O desenvolvimento de um software é composto de *ciclos*, cada um com o objetivo de gerar um entregável[3]. O RUP divide um ciclo completo de desenvolvimento em quatro fases consecutivas, cada uma concluída com um *marco*[4] bem definido. As fases são:

(i) Iniciação – durante esta fase, os casos de uso de negócio e entidades externas (atores) são levantados e o escopo do projeto é definido.

(ii) Elaboração – o propósito desta fase é fazer uma análise de domínio do problema, definir a arquitetura a ser utilizada, desenvolver o plano de projeto e eliminar os elementos de maior risco do projeto. Ao final da fase é gerado um protótipo para comprovação da arquitetura escolhida.

(iii) Construção – nesta fase, todos os demais componentes e artefatos são desenvolvidos e integrados ao produto, bem como todos os testes são realizados.

(iv) Transição – esta fase tem como propósito principal realizar a passagem do produto desenvolvido aos seus clientes.

Na Figura 2.4 podemos ver as fases descritas acima, juntamente com os marcos para fim de cada uma.

[3] Algo que provê valor aos envolvidos no processo de desenvolvimento, seja ele interno ou externo. Pode ser composto de artefatos ou de alguma saída intangível (e.g. otimização de performance).

[4] Um momento no qual certas decisões críticas são tomadas e para tal alguns objetivos devem ser atingidos.

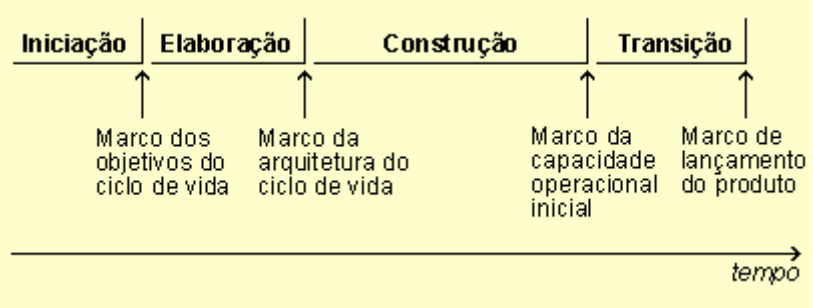

Figura 2.4: As fases e os marcos de um projeto.

O eixo vertical do gráfico, que representa o aspecto estático do processo, é composto por nove disciplinas centrais, sendo seis disciplinas de *engenharia* e três de *suporte*. As disciplinas de engenharia são:

(i) Modelagem de Negócios – descreve como desenvolver uma visão da organização e, com base nesta visão, definir os processos, os papéis e as responsabilidades dessa organização em um modelo de casos de uso de negócios e em um modelo de objetos de negócios.

(ii) Requisitos – descreve o que o sistema deve fazer de forma que os desenvolvedores e clientes entendam concordem com essa descrição.

(iii) Análise e Design – o objetivo desta disciplina é mostrar como o sistema será concebido na fase de implementação.

(iv) Implementação – o propósito da disciplina é conceber de fato o sistema, baseado nos artefatos gerados na disciplina de Análise e Design.

(v) Teste – atua em vários aspectos como uma provedora de serviços para as outras disciplinas. O teste enfatiza principalmente a avaliação da qualidade do produto.

(vi) Implantação – descreve as atividades que garantem que o produto será disponibilizado a seus usuários finais.

Já as disciplinas de suporte são:

(i) Ambiente – concentra-se nas atividades necessárias à configuração do processo para um projeto. A meta das atividades dessa disciplina é

fornecer à organização o ambiente de desenvolvimento de software (processos e ferramentas) que dará suporte à equipe de desenvolvimento.

(ii) Gerenciamento de Projeto – consiste em confrontar os objetivos da concorrência, gerenciar riscos e superar obstáculos para liberar com sucesso um produto que atenda às necessidades dos clientes e dos usuários.

(iii) Gerenciamento de Configuração e Mudança – controla mudanças feitas nos artefatos de um projeto e mantém a integridade dos mesmos.

Uma disciplina descreve todas as atividades a serem realizadas e papéis necessários para produzir um determinado conjunto de artefatos. Para exemplificar, a Figura 2.5 traz a disciplina de Modelagem de Negócios, juntamente suas atividades e os papéis responsáveis por cada uma delas.

Figura 2.5: Atividades e papéis da Modelagem de Negócios.

Como dito anteriormente, um artefato é produzido em uma determinada atividade. Os artefatos são produtos de trabalho finais ou intermediários produzidos e usados durante os projetos. São usados para capturar e transmitir informações do projeto e podem ser documentos, modelos ou elementos de modelo. Na Figura 2.6 podemos ver os principais artefatos do RUP.

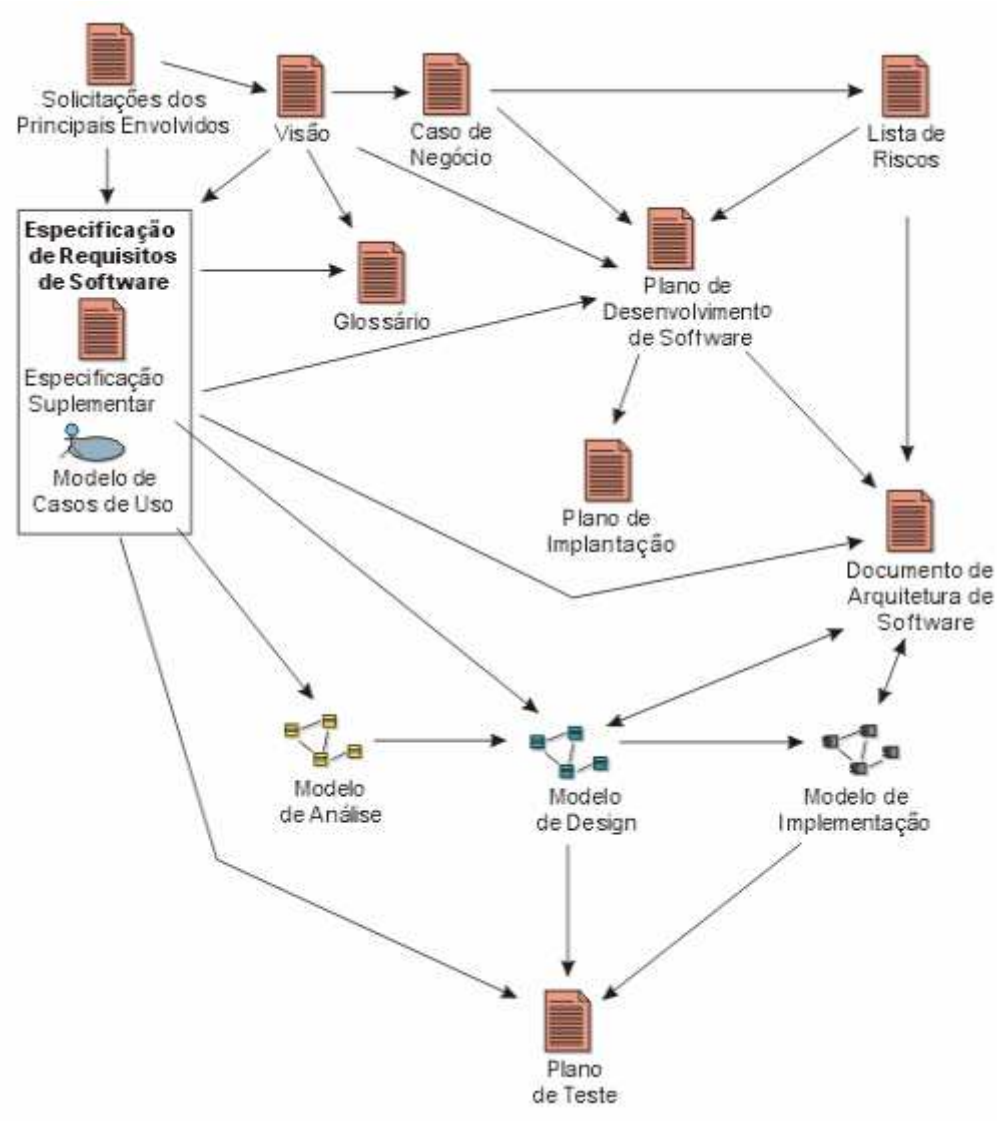

Figura 2.6: Principais artefatos do RUP e o fluxo de informações entre eles.

O processo RUP aplicado integralmente é considerado pesado e costuma ser indicado para o desenvolvimento de grandes projetos, embora seja facilmente customizável a projetos de qualquer escala. Ele contém um kit de desenvolvimento, que provê suporte para sua adequação às necessidades de uma organização. Algumas metodologias de desenvolvimento concorrentes incluem o Cleanroom (Prowell et al. 1999) e os Métodos Ágeis (Martin 2003), como a Programação Extrema (XP-Extreme Programming) (Beck e Andres 2004) e Scrum (Schwaber e Beedle 2001), dentre outros.

O RUP reúne muitas das *boas práticas* no desenvolvimento de software moderno. A seguir são apresentadas seis dessas boas práticas que alicerçam o processo:

(i) Desenvolvimento Iterativo do Software – O RUP provê uma abordagem de desenvolvimento iterativo, que possibilita o entendimento incremental do problema através de refinamentos sucessivos. Com isso, os riscos ao longo do projeto são reduzidos significativamente, uma vez que são confrontados por partes.

(ii) Gerenciamento de Requisitos – o RUP descreve como documentar funcionalidades, restrições de sistema, restrições de projeto e requisitos de negócio. Os casos de uso e os cenários são exemplos de artefatos para captura de requisitos funcionais.

(iii) Uso de Arquitetura Baseada em Componentes – o RUP oferece uma forma sistemática para construção deste tipo de sistema, focando na produção de uma arquitetura executável nas fases iniciais do projeto.

(iv) Uso de Software de Modelos Visuais – o RUP possibilita uma visão geral de uma solução ao abstrair a programação e representá-la utilizando blocos de construção gráfica. O uso de modelos visuais também permite que indíviduos de perfil menos técnico (e.g. clientes) tenham um melhor entendimento do problema.

(v) Verificação da Qualidade do Software: o RUP visa auxiliar no controle do planejamento da qualidade, verificando-a na construção de todo o processo e envolvendo a equipe de desenvolvimento.

(vi) Gestão e Controle de Mudanças do Software: o RUP define métodos para controlar e monitorar mudanças. Também define *áreas de trabalho seguras*, garantindo ao programador que mudanças feitas em outro sistema não afetarão o seu sistema.

2.4. Sistemas Multi-agentes

Desenvolver sistemas de porte industrial é uma tarefa complexa, uma vez que os mesmos possuem muitas partes e interações (Simon 1996). O ramo da Engenharia de Software tem o papel de prover estruturas e técnicas que facilitem o tratamento de tal complexidade. Para auxiliar no gerenciamento da complexidade dos sistemas, algumas técnicas são sugeridas: decomposição,

abstração e organização (Booch 1994). A decomposição é a técnica mais básica para enfrentar grandes problemas e consiste em dividi-los em pedaços menores, mais manejáveis, o que facilita o tratamento de cada um deles isoladamente. A abstração é o processo de definição simplificada de um modelo do sistema que enfatiza alguns dos detalhes ou propriedades, enquanto suprime outros. Já a organização consiste em definir e gerir as relações entre os diferentes componentes de resolução do problema. Para exemplificar tais técnicas, a Figura 2.7 exibe a visão canônica de um sistema complexo. Tal figura traz um sistema decomposto em subsistemas e componentes, juntamente com exemplos de relações possíveis entre as partes.

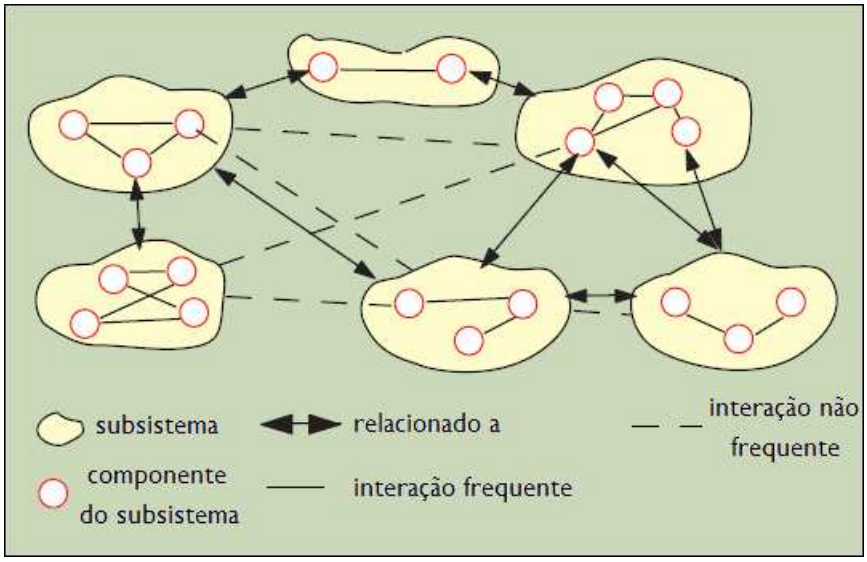

Figura 2.7: Visão canônica de um sistema complexo.

Nas últimas décadas, agentes de software têm se tornado uma abstração poderosa para dar suporte ao desenvolvimento de sistemas complexos e distribuídos (Jennings 2001). Neste contexto, inúmeros trabalhos vêm sendo propostos na literatura, como metodologias e processos (Wooldridge, Jennings e Kinny 2000) (Cossentino 2005) e linguagens de modelagem (Silva e Lucena 2007). Outras comparações sobre SMA podem ser encontradas em (Sellers e Giorgini 2005). A Figura 2.8 exibe a visão canônica de um sistema multi-agente, mostrando seus componentes, interações e o ambiente.

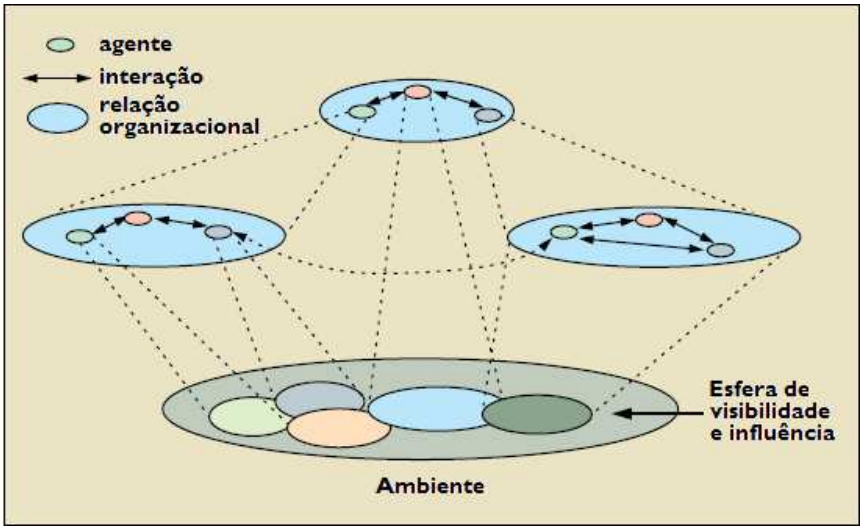

Figura 2.8: Visão canônica de um sistema multi-agente.

Um agente é um sistema encapsulado situado em algum ambiente e capaz de agir autonomicamente e de forma flexível neste ambiente a fim de atingir seus objetivos (Wooldridge 1997). Os agentes são uma metáfora natural para compreender sistemas e possuem características particulares como alta interatividade e múltiplos locos de controle. Um conjunto de agentes em um mesmo sistema forma um Sistema Muti-agente (SMA). Um agente é uma abstração que possui fundamentalmente as seguintes características:

(i) Autonomia – capacidade de agir sem intervenção externa.

(ii) Interatividade – se comunicam com outros agentes e com o ambiente por meio de mensagens e sensores.

(iii) Reatividade – capacidade de perceber o ambiente e responder às mudanças que ocorrem no mesmo.

(iv) Pró-atividade – atuam não somente em resposta ao ambiente, mas também são orientados a objetivos.

(v) Adaptação – são capazes de modificar, em algum grau, o seu comportamento devido à mudanças do ambiente e de outros agentes.

Características adicionais que podem ser presenciadas em agentes são:

(i) Aprendizado – são capazes de modificar o seu comportamento com base em sua experiência, não necessariamente relacionado às mudanças no ambiente.

(ii) Racionalidade – são capazes de selecionar suas ações com base em seus objetivos.

(iii) Mobilidade – são capazes de se mover de um ambiente para outro.

SMAs (Wooldridge 2001) sintetizam contribuições de diferentes áreas, tais como inteligência artificial, sistemas distribuídos e engenharia de software. São capazes de prover um alto grau de abstração no qual os desenvolvedores podem entender, modelar e construir sistemas complexos. No contexto da engenharia de software, eles são vistos como um paradigma que visa o desenvolvimento de sistemas que contenham muitos componentes que se interagem dinamicamente, cada um com seu próprio loco de controle. A idéia central da engenharia de software orientada a agentes é a decomposição de sistemas em entidades autônomas, pró-ativas, sensíveis ao contexto e com habilidade social, denominadas agentes. Para os SMAs, o termo autônomo consiste no fato de que os agentes têm existência própria, independentemente dos demais agentes. Normalmente, cada agente possui um conjunto de objetivos, um conjunto de capacidades comportamentais e a autonomia para utilizar tais capacidades para atingir seus objetivos.

A utilização do paradigma de SMAs para modelar e construir sistemas possui diversas vantagens, que incluem:

(i) A decomposição orientada a agentes é uma forma efetiva de particionar o escopo de problema de um sistema complexo;

(ii) a abstração principal do conceito de orientação a agentes é um meio natural de modelar sistemas complexos;

(iii) a filosofia orientada a agentes para modelar e gerenciar relações organizacionais é adequada para lidar com as dependências e interações que existem em sistemas complexos.

2.5. Linhas de Produto de Software

As iniciativas de componentização de software e de desenvolvimento orientado a objetos na década de 80 despertaram a comunidade de software para as oportunidades e vantagens da reutilização de código. O sucesso de tais atividades estimulou o surgimento de iniciativas de reuso para diversos momentos do processo de desenvolvimento de software, incluindo artefatos como documentos, especificações e modelos, o que aumentaria ainda mais a perspectiva de redução de custos e ganho de produtividade. Um dos frutos do amadurecimento de tais idéias levou a formulação do modelo de Linhas de Produto de Software (LPSs), paradigma que apresenta um deslocamento no foco tradicional de desenvolvimento de software (Durscki et al. 2004).

A idéia de linha de produtos não é nova. Há vários exemplos de utilização do conceito na história antiga, como por exemplo as pirâmides do Egito. O termo é uma clara referência às linhas de produção das indústrias de manufatura, que no final do século XIX revolucionaram o processo produtivo, sugerindo o desenvolvimento sequencial de produtos, baseado em tarefas repetitivas e executadas sempre pelas mesmas pessoas que dispunham dos recursos materiais que necessitavam.

Linhas de Produto de Software (LPSs) é uma abordagem interessante no ponto de vista do reuso de software que permite a construção sistemática de famílias de aplicações a partir da exploração das suas partes em comum. Podemos pensar em LPSs como instâncias e especializações de arquiteturas mais genéricas, uma vez que as mesmas especializam a arquitetura para um tipo específico de aplicação. O termo família de aplicações foi primeiramente introduzido por Parnas em (Parnas 1976), que definiu o mesmo como um conjunto de programas com tantas propriedades em comum que se torna vantajoso estudar as mesmas antes de analisar cada membro individualmente. O conceito atualmente dado a uma LPS é similar à definição (Clements 2002): “um conjunto intensivo de sistemas de software que compartilham e gerenciam um grupo de características comuns que satisfazem uma necessidade específica de um domínio, e que são desenvolvidos a partir de um mesmo núcleo conforme estabelecido”. Uma característica pode ser definida como uma propriedade do sistema que é relevante para algum envolvido e é utilizada para capturar semelhanças ou diferenças entre os frutos de uma linha de produto (Czarnecki e Eisenecker 2000). As características são organizadas em uma representação denominada modelo de características, que foi originalmente proposto por (Kang

et al. 1990). Tal modelo expressa as características de uma LPS como uma árvore, classificando-as entre obrigatórias, opcionais e alternativas. Características obrigatórias fazem parte do núcleo da LPS e estão presentes em todos os produtos derivados da mesma. Características opcionais estão presentes apenas em alguns membros da LPS e características alternativas são aquelas que variam de um produto a outro.

A abordagem das LPSs foca na utilização de técnicas de engenharia que permitem criar um grupo de sistemas de software similares a partir de um conjunto de especificações comuns a todos através de um meio comum de produção (Clements e Northrop 2002). Tal abordagem tem emergido rapidamente como um paradigma de desenvolvimento de software porque permite criar uma estrutura reusável e de rápida customização. A engenharia de LPS é tipicamente composta de dois processos-chave (Pohl, Bockle e Van Der Linden 2005): engenharia do domínio, no qual as características comuns e variáveis são identificadas, definidas e realizadas, e engenharia de aplicação, no qual aplicações da LPS são construídas através do reuso dos artefatos do domínio e a exploração da variabilidade da LPS. Apesar dessa abordagem envolver muito reuso, o mesmo é feito de forma planejada, uma vez que toda característica identificada é projetada e implementada levando em consideração a possibilidade de reuso e modificação para se enquadrar nos vários sistemas contemplados pela LPS.

As LPSs são projetadas para reconfiguração. Tal reconfiguração pode envolver a adição ou remoção de componentes do sistema, definição de parâmetros e restrições para os componentes, além da inclusão de conhecimento dos processos de negócio. Podem ser configuradas em dois pontos no processo de desenvolvimento:

(i) Configuração em Tempo de Implantação – o sistema é projetado para ser configurado pelos clientes. O conhecimento dos requisitos e do ambiente operacional é incorporado em um conjunto de arquivos de configuração usados pelo sistema genérico.

(ii) Configuração em Tempo de Projeto – a organização responsável pelo desenvolvimento do software projeta-o a partir de um núcleo comum da LPS e trabalha sobre a especificidade do cliente, o que gera um novo sistema.

Existem diversas motivações para a adoção da abordagem de LPSs, sendo as principais a redução de custos no desenvolvimento, reforço da qualidade e redução no tempo de entrega da solução. O reuso de artefatos para derivar produtos da LPS é responsável pela redução nos custos e no tempo de entrega. Já o reforço da qualidade é proveniente da revisão e teste dos artefatos da LPS em diversos produtos. O desenvolvimento de artefatos reusáveis requerem um investimento e tempo maiores na fase inicial de construção da LPS, mas na maioria dos casos tais esforços são compensados a partir da terceira derivação (Pohl, Bockle e Van Der Linden 2005). Outras motivações para o uso de LPSs são a redução no esforço de manutenção, evolução organizada e redução na complexidade. A redução no esforço de manutenção se dá porque sempre que um artefato da linha de produto é modificado, a mudança é propagada para todos os produtos que utilizam tal artefato. A introdução de um novo artefato no núcleo da LPS possibilita a evolução organizada de todos os tipos de produtos derivados. Já a redução da complexidade ocorre porque LPSs fornecem uma estrutura que determina quais componentes podem ser reutilizados em quais lugares através da definição de variabilidade em locais distintos. Há ainda uma classificação dos benefícios feita por (Cohen 2003) para mapear as vantagens propostas pelo modelo:

(i) Tangíveis – benefícios que podem ser medidos diretamente, como por exemplo redução do tempo de entrega ou redução de defeitos.

(ii) Intangíveis: benefícios que os desenvolvedores relatam, mas que não podem ser medidos com métricas. Tais benefícios podem incluir satisfação do cliente e ânimo dos recursos, dentre outros.

Existem diversas metodologias de LPS propostas, como por exemplo FORM (Kang et al. 1998), KobrA (Atkinson, Bayer e Muthig 2000) e PLUS (Gomaa 2004). Este trabalho foi desenvolvido com base na metodologia PLUS, que é abordada de forma mais elaborada a seguir.

2.5.1. Product Line UML-based Software Engineering (PLUS)

Product Line UML-based Software Engineering (PLUS) (Gomaa 2004) é uma metodologia para projetar LPSs baseada na notação UML, que provê uma série de conceitos e técnicas para estender às mesmas os modelos e processos

tradicionais de sistemas únicos (e. g. casos de uso). A metodologia é constituída de dois processos principais: Engenharia de LPS e Engenharia de Aplicação. Cada um dos processos é composto de cinco fases: Requisitos, Análise, Design, Implementação e Teste. Na Engenharia de LPS, as partes comuns e variáveis são analisadas de acordo com os requisitos gerais da LPS. São desenvolvidos o modelo de casos de uso, o modelo de análise, arquitetura e componentes reusáveis da LPS. Todos os artefatos produzidos em cada uma das fases são armazenados em um repositório. Já na Engenharia de Aplicação ocorre a derivação das aplicações individuais, com base nos artefatos armazenados no repositório. A Figura 2.9 traz a estrutura da metodologia PLUS, com seus processos, repositório, atores e as interações entre os mesmos.

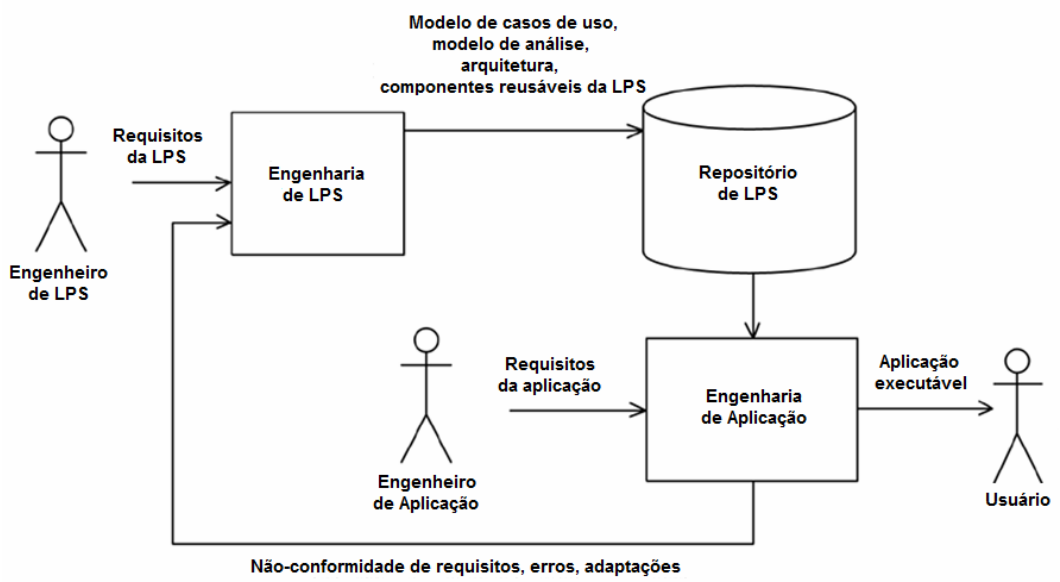

Figura 2.9: Estrutura da metodologia PLUS.

A metodologia PLUS fornece uma notação baseada na UML para expressar variabilidade nos modelos de requisitos, análise e design. Tal notação utiliza estereótipos para diferenciar os elementos do modelo, como <<*kernel*>>, <<*optional*>> e <<*alternative*>> para representar a categoria de reuso; e <<*entity*>> e <<*control*>> para representar o papel da classe na aplicação. PLUS ainda propõe padrões arquiteturais para LPSs e notações, como por exemplo o uso de pacotes ou tabelas para representar elementos ou características das dependências dos modelos.

2.6. Linhas de Produto de Sistemas Multi-agentes

Ao lidar com sistemas complexos, em particular sistemas que exibem algum tipo de autonomia ou propriedades autonômicas, é comum observar que o sistema não permanece estático. Sistemas complexos evoluem ao longo do tempo, e a arquitetura de um sistema em desenvolvimento sofre modificações mesmo em tempo de execução. Um sistema em evolução pode ser visto como uma versão do sistema em questão. Isto é, à medida que o sistema evolui, o mesmo passa a representar uma nova instância de si mesmo, com suas próprias variantes e mudanças específicas. Em outras palavras, um sistema evolutivo pode ser visto como uma linha de produtos, no qual a arquitetura central da LPS é fixa e cada versão desse sistema pode ser vista como um produto particular dessa LPS. Ao considerarmos a parte imutável de um sistema como sendo uma arquitetura central, a especialização em variados produtos pode ser vista como adições baseadas em agentes. Dessa forma, podemos intuir que um sistema evolutivo pode ser encarado como uma Linha de Produto de Software de Sistemas Multi-agentes (Peña et al. 2006b).

A combinação de LPS e SMA é denominada Linhas de Produto de Sistemas Multi-agentes (LP-SMA) e visa incorporar os benefícios de cada uma das partes e aplicar na indústria a tecnologia de agentes (Nunes 2009). Tal combinação é recente na literatura e ainda apresenta muitos desafios a serem superados, como é apresentado em (Peña, Hinchey e Ruiz-Cortéz 2006). Foram propostas abordagens para a composição do núcleo de uma LP-SMA (Peña et al. 2006a), captura de variabilidade na fase de análise (Dehlinger e Lutz 2005), processos (Nunes et al. 2009b) e estudos empíricos (Nunes et al. 2009c). Por um lado, as LPSs cobrem todo o ciclo de desenvolvimento de uma família de produtos de forma a tornar a derivação de produtos concretos rápida e sistemática. Por outro lado, a Engenharia de Software Orientada a Agentes (ESOA) é um novo paradigma da engenharia de software que surgiu com o propósito de fornecer melhores práticas no desenvolvimento de sistemas focando no uso de agentes como principal abstração.

2.7. Considerações Finais

Este capítulo apresentou uma visão geral sobre Gerenciamento de Projetos, a metodologia de desenvolvimento Rational Unified Process, Sistemas Multi-agentes, Linhas de Produto de Software e a junção destes dois últimos

temas, as Linhas de Produto de Sistemas Multi-agentes. O gerenciamento de projetos consiste em coordenar atividades com o objetivo de atingir as expectativas dos envolvidos, mantendo os riscos de fracasso em um nível tão baixo quanto necessário durante o ciclo de vida de um projeto. O RUP é um processo de desenvolvimento de software que fornece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades aos integrantes de uma organização de desenvolvimento. As LPS permitem o desenvolvimento de famílias de aplicações que compartilham características comuns e variáveis através de um método sistemático. Já os SMAs são sistemas cuja arquitetura é composta por agentes de software, que são uma poderosa abstração para desenvolver sistemas complexos e distribuídos. Diversos trabalhos foram propostos nas áreas de SMA e LPS no intuito de prover técnicas como processos, metodologias e linguagens de modelagem para auxiliar no desenvolvimento de sistemas. A junção de LPSs e SMAs visa incorporar os benefícios de cada uma das partes e aplicar na indústria a tecnologia de agentes.