Vol.2010-SE-167 No.25
2010/3/19

組込みソフトウェアの仕様モデルとアーキテクチャの関係に関する考察

加藤大地[†1]　沢田篤史[†2]
張漢明[†2]　野呂昌満[†2]

PLSE 開発においては，アーキテクチャを特定しなければアプリケーション工学の工程を開始できないことから，アーキテクチャ定義の自動化支援のために仕様モデルとソフトウェアアーキテクチャ間の関係における追跡性を保証することはとくに重要である．我々は，仕様モデルの構成要素に型を導入しその型と構成要素間の関係との組合せパターンがアーキテクチャ上のパターンと構文的に対応可能であることを確認した．型を導入した拡張フィーチャモデルと我々が提案するアスペクト指向ソフトウェアアーキテクチャスタイル E-AoSAS++を用いて定義したアーキテクチャ上のアスペクトの対応関係について議論する．

Issues on Mapping between Specification Model Software Architecture in Embeded Systems

DAICHI KATO,[†1] ATSUSHI SAWADA,[†2]
CHANG HAN-MYUNG[†2] and MASAMI NORO[†2]

Since software development based on PLSE does not stand without software architecture, it is especially important to define the traceability between a specification model and software architecture for the purpose of automated architecture definition. We have realized that a pattern among components in a specification model could be syntactically mapped to another pattern among components in an architecture if we could put a designated color on a set of components in a specificaiton model. The paper concerns the mapping between the pattern in the extended feature model where types for sets of features are introduced and the aspect on the architecture in E-AoSAS++ which is an aspect-oriented software architecture style we have defined.

1. はじめに

Product Line Software Engineering(PLSE)[1] は共通のソフトウェアアーキテクチャを持つ製品系列においてソフトウェア部品をコア資産として再利用することで生産性を向上させる技術である．コア資産中のソフトウェア部品としては，仕様モデル，アプリケーションフレームワーク，コンポーネント仕様とコンポーネント (プログラムコード) などが挙げられる．PLSE ではアーキテクチャがすべての基礎なので，アーキテクチャとその他のコア資産の追跡性を明確にすれば，PLSE が本来目指すプログラミングレス開発または自動プログラミングの支援が可能になる．

我々は，組込みシステムのためのアスペクト指向ソフトウェアアーキテクチャスタイル，E-AoSAS++[2] を提案している．E-AoSAS++に基づく PLSE 開発において，プログラミングレスを実現することを我々は目的としている．このことから，E-AoSAS++で記述したアーキテクチャとコア資産の追跡性を保証することは，我々にとっても，きわめて重要な問題である．

コア資産とアーキテクチャの関係では，とりわけ仕様モデルとの追跡性を明らかにすることが重要であると我々は考えている．PLSE のアプリケーション工学はソフトウェアアーキテクチャを特定してから開始され，アプリケーション工学への入力となるソフトウェアアーキテクチャはドメイン工学段階で要求を分析しながら，仕様モデルをアプリケーションに向けて洗練することで定義する．我々は，PLSE 研究の原点の 1 つともいえる FORM[3] で提案されているフィーチャモデルを，広く一般に用いられているとの認識から，仕様モデル記述に用いる．E-AoSAS++で記述したアーキテクチャとフィーチャモデルで記述した仕様モデルとの追跡性を確保することが本研究の目的である．

本来，仕様とアーキテクチャの関係は，アーキテクチャ設計における，要求分析の段階で定義されるものと考えるのが自然である．i*フレームワーク[4] や KAOS[5] などのゴール指向分析手法とアーキテクチャ上のアスペクトの関連を考察したさいに，仕様モデルとアーキテクチャとの間の追跡性を構文的に保証するための着想を得た．すなわち，仕様モデルの構成要素の種類と構成要素間の関係の組合せパターンから，アーキテクチャの構成要素と構成

†1 南山大学大学院数理情報研究科
Graduate School of Mathematical Sciences and Informaition Enginnering,Nanzan University
†2 南山大学数理情報学部
Faculty of Mathematical Sciences and Information Engineering,Nanzan University

要素間の関係の組合せパターンへの対応関係が存在することを確認した．このさい，デザインパターン[6] を横断的コンサーンの分離を意図したマイクロアーキテクチャであると捉え，対応関係の考察を行った．結果として，仕様モデルの構成要素を分類し型を定義し，仕様モデル上の構成要素間の関係とその型との組合せパターンで，アーキテクチャ上のアスペクトの記述パターンが特定できる場合があることを確認した．

本研究では，フィーチャに型を導入し，フィーチャとそれらの関係との組合せパターンとアーキテクチャ上のアスペクト記述パターンとの関係を定義し提案する．本来，意味の領域に踏み込んではじめて明らかになる関係が，フィーチャの型を導入することにより，構文的に定義可能なことを示す．

2. E-AoSAS++

E-AoSAS++は，我々が提案するソフトウェアアーキテクチャスタイルであり，特に組込みソフトウェアのアーキテクチャを記述することを目的としている．組込みソフトウェアには，実時間処理や並行処理，耐故障性などのコンサーンがハードウェア制御のためのモジュール分割中に横断的に存在する．これらのコンサーンとコアコンサーンであるハードウェア制御を矛盾なくモジュール化するために，E-AoSAS++はアスペクト指向に基づくものとして定義した．

E-AoSAS++では，ソフトウェアを並行に動作する状態遷移機械（CSTM）の集合であると規定し，コアコンサーンのハードウェア制御をモジュール化する．アスペクトは並行に動作する状態遷移機械 (CSTM) の集合であり，アスペクト間記述はメタ状態遷移機械として記述する．こうすることにより，組込みソフトウェア記述において従来から使用されてきた標準的な記述方法である状態遷移でアーキテクチャ記述を統一することを可能にした．UML を拡張したアーキテクチャ記法により，複数の CSTM 間の階層的な包含関係，依存関係，CSTM の振る舞いや通信方法，計算実体への割当を表現する．記述される観点と表現に用いる UML 図の関係は次の通りである．

表 1 に示した UML 図を用いて 3 種類のアーキテクチャを記述する．以下に E-AoSAS++ に基づく開発体系で設計するアーキテクチャを示す．

- コンセプチュアルアーキテクチャ
  システムに横断する関心事を抽出したアーキテクチャ．関心事（VIew）毎に記述する
- プロダクトラインアーキテクチャ
  製品系列全体の構成を表すアーキテクチャ
- プロダクトアーキテクチャ
  プロダクトラインアーキテクチャを基にプロダクトに必要なコンポーネントを抽出し，プロダクト毎に作成するアーキテクチャ

各アーキテクチャと取り扱うモデル，図式表現の関係は図 1 の通りである．

表 1 記述の観点と用いる UML 図の関係
Table 1 Relation between architechture view and UML diagram.

観点	UML 図
階層的な包含関係	コンポーネント図，クラス図
モジュールの振る舞い	状態マシン図
通信方法	シーケンス図
計算実体への割当	オブジェクト図

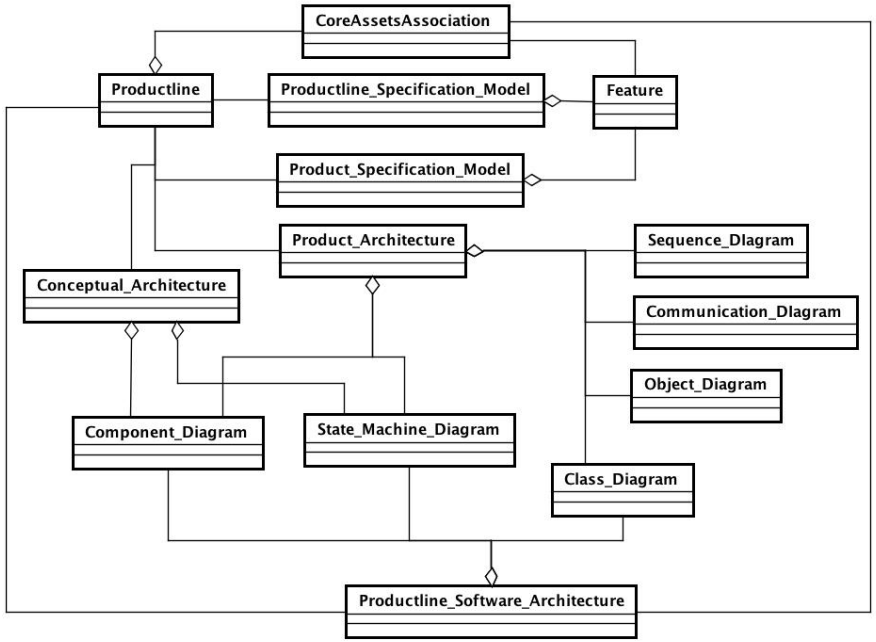

図 1 E-AoSAS++の開発体系におけるアーキテクチャと取り扱うモデル，要素の関係

Vol.2010-SE-167 No.25
2010/3/19

3. フィーチャモデルの拡張

FORM によるフィーチャモデルは，1 つの分割方針に従っている．これでは E-AoSAS++ の利点である横断的関心事の表現との対応関係を構築することは本質的に難しい．そこでフィーチャモデルの表現に型を導入することによって拡張する．

3.1 FORM

フィーチャモデルとは，ユーザ要求を基に製品間の共通性と相違性の整理を目的とした仕様モデルである．フィーチャとは外部から認識できる製品の特徴または機能である．FORM はフィーチャを次の 4 つのレイヤに分類する．

- Capability Layer
  ユーザの要求する機能，ユーザ要求を実現する製品の機能，非機能
- Operating Environment Layer
  ハードウェア，ソフトウェアの操作環境
- Domain Technology Layer
  特定のドメイン内で使用する技術
- Implementation Technique Layer
  複数のドメインで使用する一般的な技術

フィーチャは製品系列で必須 (Mandatory)，任意 (Optional)，選択 (Alternative) かに分類される．フィーチャ間の関係は Composed-of(全体-部分関係)，Generalization/Specialization(汎化/特化関係)，Implemented-by(実現関係) の 3 種類で木構造を構成し表現する．凡例を図 2 に示す．

3.2 型定義による意味の拡張

横断的関心事を表現できるアスペクト指向の利点を生かして対応関係を構築するためには，フィーチャの意味を拡張するための型を導入する必要があると考える．本稿では，一般的に用いられるアーキテクチャが示す関心事を，デザインパターンから抽出し，分類した．分類する際に，どのような型を定義すればよいか検討し，次のような型を定義した．

- normal
  標準のフィーチャを示す型
- input
  入力処理を必要とするフィーチャを示す型
- output

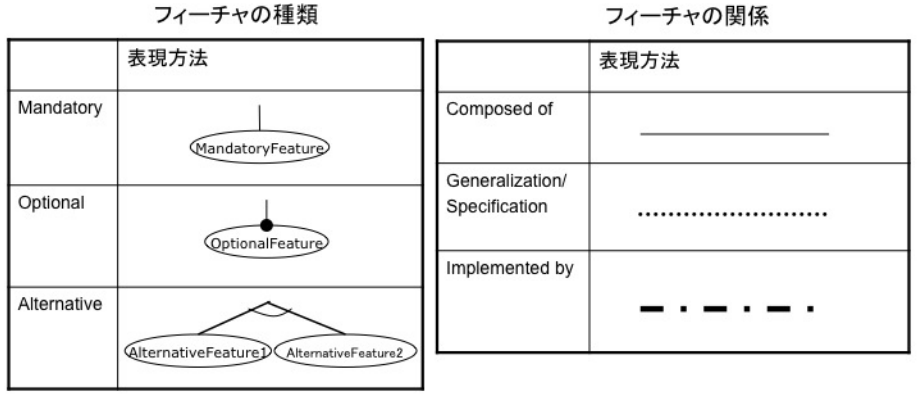

図 2 フィーチャダイアグラムの凡例

出力処理を必要とするフィーチャを示す型

- configuration
  自身がシステムの構成の 1 つとして実現されるフィーチャの型

input，output，configuration 型に関してはフィーチャダイアグラム上にステレオタイプとして記述する．

アーキテクチャとの関係が，フィーチャダイアグラムにおいてどのような構造的関係を表す構文で表されるのかを整理するために，フィーチャ構造ごとにグループ化を行った．表 2 に分類したグループの名前と，各グループが表現するフィーチャの構造的な関係を示す．

表 2 グループの名前とフィーチャ間の構造的関係
Table 2 Group name and structural relationship among features.

グループの名前	フィーチャの構造的関係
Composed	Capability Layer に配置され，全体/部分関係となるフィーチャ群
Generalization/Specialization	Capability Layer に配置され，汎化/特化関係となるフィーチャ群
Primitive	Capability Layer に配置される個々のフィーチャ
Implemented	Capability Layer と Operating Environment Layer 間において実現関係にあるフィーチャ群
Strategy	Domain Technology Layer または Implementation Technique Layer に配置され，汎化/特化関係かつ選択関係にあるフィーチャ群

Vol.2010-SE-167 No.25
2010/3/19

次に，横断的関心事がフィーチャもしくはフィーチャ群では，どのような対応関係で示されるのかをフィーチャ構造のグループ毎に示す．

Composed

- Facade 関係
  normal 型のフィーチャで，複数のサブフィーチャを集約している場合，複数の類似機能を集約した Aspect と対応する
- Observer 関係
  input 型，output 型のサブフィーチャがそれぞれが最低１つ以上含まれて集約されるフィーチャ群が View と対応し，サブフィーチャは入力に関する Aspect，出力に関する Aspect，それ以外の Aspect でそれぞれ実現されるという関係になる
- Mediator 関係
  configuration 型のフィーチャを，サブフィーチャとして集約しているフィーチャが存在する場合，そのフィーチャに対応する View が存在する。

Generalization/Specialization

- Template Method 関係
  複数のサブフィーチャによって構成され，親フィーチャとの関係が汎化／特化の関係にある場合，それぞれのサブフィーチャは Aspect として実現される

Primitive

- Configuration 関係
  configuration 型のフィーチャはシステムの構成の１つとして対応するので，状態として表現される可能性がある．

Implemented

- Bridge 関係
  実現関係で結ばれているフィーチャの組 (CapabilityLayer のフィーチャと OperatingEnvironmentLayer のフィーチャ) は，CapabilityLayer のフィーチャが実現されている Aspect 内に OperatingEnvironmentLayer 内のフィーチャが実現された PrimitiveCSTM が存在する

以上に定義した関係を用いてアーキテクチャと仕様モデルの関係を構築する．

4. 対応関係の例

4.1 対象のドメイン

本節では，提案する対応づけ指針の妥当性を評価するために，組込みソフトウェアとして代表的な自動販売機制御ソフトウェアを挙げる．

4.1.1 自動販売機制御ソフトウェアのプロダクトラインフィーチャダイアグラム

図３に自動販売機制御ソフトウェアのプロダクトラインフィーチャダイアグラムを示す．この例で示す自動販売機制御ソフトウェアは，販売とメンテナンスの２つのモードを持つ一般的な自動販売機の制御ソフトウェアの例である．

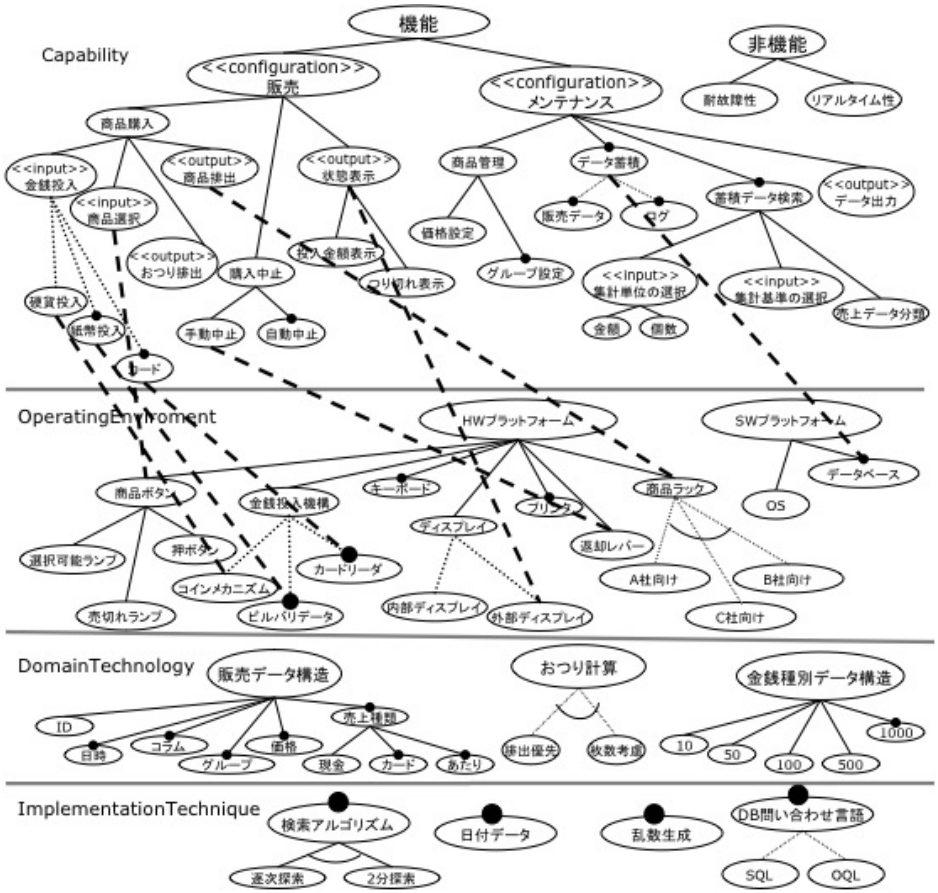

図 3 自動販売機制御ソフトウェアのプロダクトラインフィーチャダイアグラム

4.1.2 商品購入フィーチャとアーキテクチャの Observer パターン関係

商品購入フィーチャとアーキテクチャの Observer パターン関係を図 4 に示す．商品購入フィーチャのサブフィーチャは，Observer 関係を示すフィーチャである．4つのサブフィーチャで構成され，input 型のフィーチャと output 型のフィーチャを含んでいるので，商品購入と対応した View，各サブフィーチャが input 型，output 型に対応した Aspect と対応している．この例では，全てのサブフィーチャが Aspect と 1 対 1 で対応しているが，入力に関する Aspect として Money_Injection と Item_Selection，出力に関する Aspect として Item_Ejection，Change_Return として対応しており，入力に関する Aspect と出力に関する Aspect との対応関係が構築できる．他の場合でも，入力に関する Aspect と出力に関する Aspect を特定すれば対応関係が構築できると考えられる．

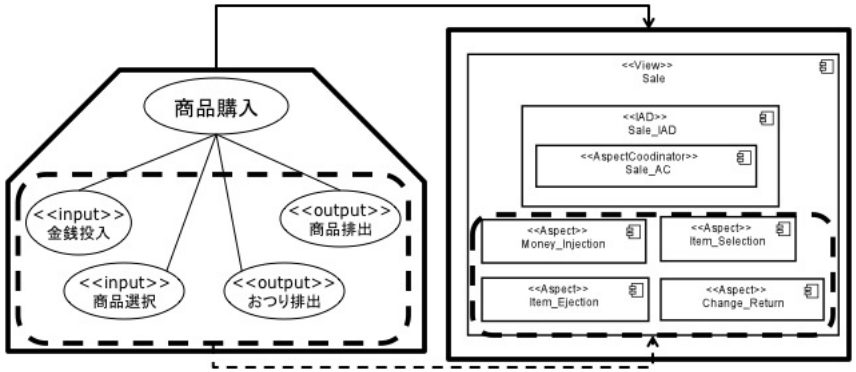

図 4 販売フィーチャとアーキテクチャとの Observer 関係

4.1.3 金銭投入フィーチャとアーキテクチャの TemplateMethod 関係

金銭投入フィーチャとアーキテクチャの TemplateMethod 関係を図 5 に示す．金銭投入フィーチャは，複数の金銭投入方法を汎化したフィーチャである．サブフィーチャが取り扱う金銭によって特化されているので，TemplateMethod 関係であると考えられる．それぞれのサブフィーチャは各 Aspect と 1 対 1 で対応するので，同様の形であれば他の例でも対応関係を構築できると考えられる．

4.1.4 実現関係をもつフィーチャ群とアーキテクチャの Bridge 関係

実現関係をもつフィーチャ群とアーキテクチャの Bridge 関係を図 6 に示す．Capabiility-

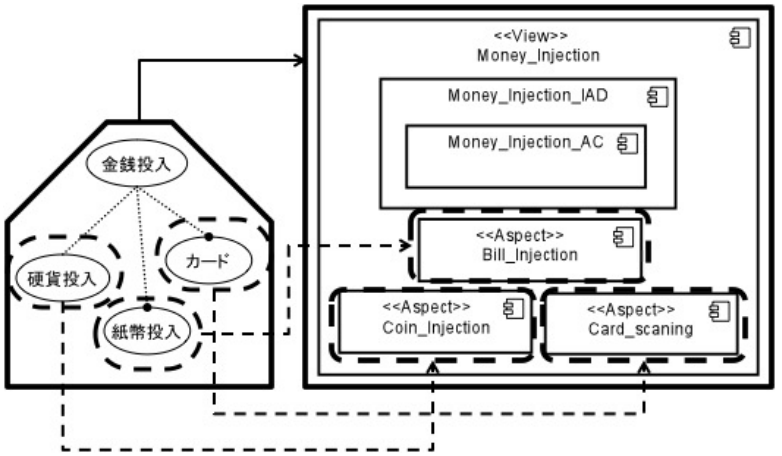

図 5 金銭投入フィーチャとアーキテクチャの TemplateMethod 関係

Layter の硬貨投入フィーチャ，紙幣投入フィーチャ，カードフィーチャは，OperatingEnviromentLayer のコインメカニズムフィーチャ，ビルバリデータフィーチャ，カードリーダフィーチャとそれぞれ実現関係を持っており，CapabiilityLayter のフィーチャは Aspect として実現され，OperatingEnviromentLayer のフィーチャは PrimitiveCSTM として実現されるという関係になる．

5. ま と め

本稿では，組込みソフトウェアの仕様モデルとアーキテクチャの関係を明確化するために，仕様モデルであるフィーチャモデルに型を導入し，フィーチャとそれらの関係との組合せパターンとアーキテクチャ上のアスペクト記述パターンとの関係について提案を行った．自動販売機制御ソフトウェアの事例を用い，対応関係の具体的な利用方法を示すとともにその妥当性についての確認を行った．その結果，我々の提案する対応関係が一般的なドメインにおいても適用可能であることが示唆された．

今後の課題として，事例検証を重ね，他に定義するべきフィーチャの型の必要性について検討すること，対応関係を用いた支援環境構築の可能性について検討することなどが挙げられる．

謝辞 本研究の一部は，科学研究費補助金（基盤研究（C）21500042）および南山大学

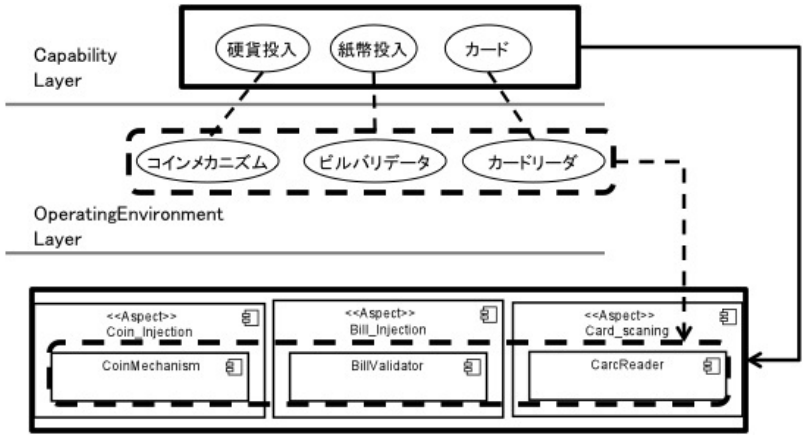

図 6 実現関係をもつ硬貨投入，紙幣投入，カードフィーチャとアーキテクチャの Bridge 関係

パッヘ奨励金 I-A-1（平成 21 年度）の助成による．

参 考 文 献

1) K.Pohl, G.Bockle, and F.Linden, *Software Product Line Enginnering: Foundations, Principles and Techniques* , Springer, 2005.
2) 加藤大地, 蜂巣吉成, 沢田篤史, 野呂昌満, “アスペクト指向に基づくソフトウェアアーキテクチャの文書化方式, ” 知能ソフトウェア工学研究会 *(KBSE)*, vol.108, no.449, pp55-60, 2009.
3) K.C.Kang, S.Kim, J.Lee, K.Kim, G.J.Kim, and E.Shin, “FORM:A Feature-Oriented Reuse Method with Domain-Specific Reference Architechtures, ” *Annals of Software Engineering*, vol.5, no.1, pp.143-168, 1998.
4) University of Toronto, i*homepage, http://www.cs.toronto.edu/km/istar/, 2003
5) E.Letier, *Reasoning about Agents in Goal-Oriented Requirements Engineering*, Universite Catholique de Louvain, Dept. Ingenierie Informatique, 2001.
6) E.Gamma, R.Helm, R.Johnson, and J.Vlissides,*Design Patterns: Elements of Reusable Object-Oriented Software*,Addison-Wealey,1995.