소프트웨어 프로덕트 라인에서의 성적관리 컴포넌트 모델링에 관한 연구

김수연°, 김지영, 김행곤
대구가톨릭대학교 컴퓨터공학과
e-mail:{gjtns9384°, kimjy, hangkon}@cu.ac.kr

A Study on Modeling of Test Result Management Component based on Software Product Line

Su-Youn Kim°, Ji-Young Kim, Haeng-Kon Kim
Dept. of Computer Engineering, Catholic University of Daegu

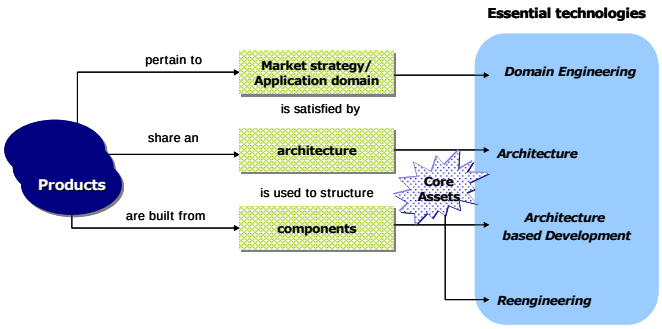
요 약

프로덕트 라인은 다양하고 빠르게 변화하는 시장의 요구사항과 특정 도메인 영역에 속하는 애플리케이션 간의 재사용 가능한 아키텍처 및 컴포넌트의 구성으로부터 연관된 시스템 구축 시 생산성과 품질의 향상을 제공함으로써 현재 많은 관심의 초점이 되고 있다. 컴포넌트의 가변성은 컴포넌트를 사용자의 요구사항에 알맞게 기능을 특화할 수 있다. 또한 프로덕트 라인에서 컴포넌트 내부에 공통으로 사용할 워크플로를 가지고 있어 컴포넌트 사용자는 순차 다이어그램 등을 통해 메시지 흐름을 직접 구현할 필요가 없고, 재사용 가능한 아키텍처는 많은 변화 계획들과 메카니즘을 포함하고 있다. 하지만, 아키텍처를 설계하기 위한 아키텍처에서의 변화성 관리와 컴포넌트의 변화성에 대한 명확한 방법이 미흡하다. 따라서 본 논문에서는 재사용 가능한 아키텍처를 설계하기 위해 변화성의 명확한 표현과 컴포넌트의 변화성을 설계하기 위해 다중 뷰의 모델링을 통하여 프로덕트 라인의 다양한 측면을 제시하고, 사례연구로 성적관리 컴포넌트 모델링에 적용해 보고자 한다.

키워드 : 프로덕트라인, 아키텍처 변화성, 컴포넌트 변화성, 성적관리 컴포넌트, 모델링

1. 서론

프로덕트 라인은 컴포넌트 기반 개발 보다 보다 효율적이고 범위가 큰 재사용을 제공하는 프레임웍 기반으로 연관된 시스템 그룹의 아키텍처를 기반으로 공통성과 변화성 부분으로 구성되고, 변화성 부분은 수정되거나 변경될 수 있다. 또한 프로덕트라인 개발의 생산성, 비용 및 품질 향상을 위해 재사용 가능한 컴포넌트들을 공유함으로써 최상의 애플리케이션 개발을 위한 패러다임으로 인식되고 있다. 프로덕트 라인 개발 방법에서는 일반성과 변화성을 포함하는 객체지향 도메인 모델을 사용하여 다양한 관점들의 메타 모델링으로의 접근이 가능하다[1,2]. 메타 모델은 생명주기 단계와, 단계내의 뷰, 각각 뷰의 메타 클래스, 뷰들간의 관련성 등을 나타내고 있다. 일반성과 변화성의 모델링 방법인 다중 뷰 모델링 방법을 이용하여 뷰들이 서로 어떻게 연관되는지, 그리고 변화성이 다른 뷰의 변화성과 어떻게 관련되는가를 제시한다. 또한 기능적인 뷰는 요구사항 분석 단계에서 유스케이스 모델을 통해 표현되고, 정적인 뷰는 클래스 모델을 통해서 표현되며, 동적인 뷰는 상태 모델과 협동모델을 통해 표현된다. 컴포넌트는 전체 시스템에서 하나의 부품 역할을 하여 다른 컴포넌트와 조립을 통해 기능을 제공하게 되는데 하나의 컴포넌트가 다른 컴포넌트와 상호작용을 할 때 불일치 문제가 발생하게 된다. 즉, 프로덕트 라인에서는 컴포넌트가 서로 잘 맞물려 작동하지 않은 경우를 말하는데 이러한 불일치 문제를 해결하기 위해 도메인별 패밀리 멤버간에 공통적인 휘처에 대해 아키텍처를 기반으로 어떠한 컴포넌트가 존재하게 되고 이들 사이의 상호작용이 어떻게 이루어지는지에 대한 기본 구조를 프레임웍 형태로 제공한다[3,4]. 이 휘처 모델링을 기반으로 하여 추출해낸 재사용성 있는 소프트웨어는 시스템 개발 초기부터 재사용성을 고려하여 생산되었기 때문에, 분석 대상 영역에서 변화 가능한 많은 응용 시스템에 대하여 적응성이 높고, 분석 단계 모델을 통해 재사용될 수 있다는 점에서 매우 가치가 높다고 할 수 있다. 그러나 지금까지 이러한 변화들이 일어나는 상황을 이해하는 것과 특별한 상황에서도 가능하게 하는 옵션들을 기록하는 것은 명확히 이루어지지 못하였다. 또한, 아키텍처가 오랜 기간 동안 많은 프로덕트 버전에서 사용되어 진다거나, 다른 프로덕트들의 설계를 위해 사용되어 지는



(그림 1) 소프트웨어 프로덕트 라인

아키텍처에서의 프로덕트 라인 문맥에서라면, 매우 중요하게 다루어진다. 따라서 본 논문에서는 재사용 가능한 아키텍처를 설계하기 위해 변화성의 명확한 표현과 컴포넌트의 변화성을 설계하기 위해 다중 뷰의 모델링을 통하여 프로덕트 라인의 다양한 측면을 제시하고, 성적관리 컴포넌트 모델링을 사례연구로 적용해 보고자 한다.

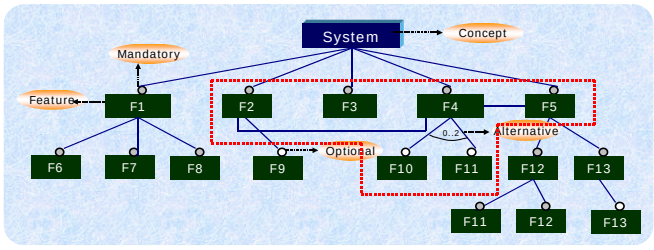
2. 관련연구

2.1 소프트웨어 프로덕트 라인

소프트웨어 프로덕트 라인은 공통성을 공유하는 소프트웨어 집약적인 시스템 집합이고, 또한 특정 시장 부문의 필요성을 만족하고, 공통 핵심자산 집합으로부터 개발되어진 특징들의 집합이다. 즉 프로덕트는 시장 전략과 애플리케이션 도메인에 적합하고, 아키텍처를 공유하고 기존 컴포넌트로부터 구축되어진다. 프로덕트 라인 아키텍처는 연관된 프로덕트 집합과 조직에 의해 개발된 시스템을 위한 공통 아키텍처로 향상된 생산성과, 소프트웨어 품질을 제공한다. 프로덕트 라인과 관련된 연구는 다음과 같이 카테고리 될 수 있다: 아키텍처 정의 및 전개, 컴포넌트 개발, COTS 활용, 기존 자산 마이닝, 소프트웨어 시스템 통합. 이들은 실제 핵심 자산과 프로덕트를 생성하고 전개하는 적절한 기술 적용에 대한 필요성이다. 프로덕트 라인 아키텍처는 특정 시스템 시장을 목표하고 있는 시스템이며 이 시스템 패밀리를 위한 기반이 될 수 있고 재사용 가능한 모델이며 공통 자산을 통해 만들어진다. 컴포넌트가 플러그인 될 수 있는 프레임웍을 제공하는 아키텍처를 기반으로 필요한 컴포넌트를 선택적으로 조립함으로써 시장 요구에 맞는 시스템을 생산해 간다[5,6].

2.2 프로덕트 라인의 변화성

동일한 프로덕트 군에 속하는 프로덕트들은 많은 공통성을 가지지만, 또한 프로덕트 사이의 변화성도 있다. 변화성은 다양한 사용자, 다양한 설계와 구현요구사항에 따라 제공되고, 도메인 분석 시에 식별된다. 따라서 변화성은 프로덕트 라인 아키텍처 설계 단계에서 고려되어지고, 코드 레벨이 아닌 아키텍처레벨에서 다루어져야 한다. 또한 변화성은 컴포넌트 조립 시에 컴포넌트 행위가 변경될 수 있는 특정 변화점에서 가능하고, 컴포넌트 설계 동안에 결정된다. 변화성은 다음과 같은 범주로 분류된다. 휘처 변화성은 특정 휘처의 정의와 구현에서 변화성, 하드웨어 플랫폼 변화성은 제어기, 메모리, 장비 등의 타입에서 변화성, 성능과 속성



(그림 2) 다이어그램으로 표현된 휘처모델

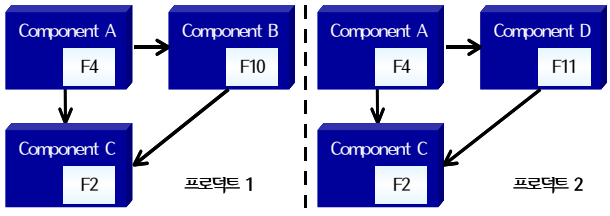
변화성은 요구된 성능과 동시에 지원 및 오류 관리와 같은 속성에서의 변화성을 의미한다[4,5].

3. 프로덕트라인 아키텍처에서의 컴포넌트 변화성

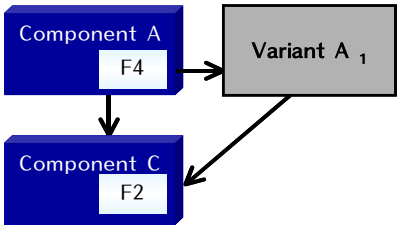
프레임웍은 아키텍처, 인터페이스, 컴포넌트, 컴포넌트간의 관계 그리고 커넥터로 구성된다. 한 프레임웍은 응용시스템이나 서브시스템 개발에 필요한 여러 컴포넌트들을 모델링 하여 이를 포함한다. 프레임웍은 동일 도메인내의 여러 응용시스템에 공통성을 기반으로 만들어지며 가변성이 제공되므로 여러 응용시스템에 재사용 가능하다. 프레임웍은 그 내부에 응용시스템들이 공통적으로 포함하고 있어야 하는 전체적인 계층구조와 계층별 컴포넌트들의 상호작용 규칙이 정의된다. 컴포넌트들 간의 상호작용은 컴포넌트가 가지고 있는 인터페이스를 사용한다. 커넥터는 호출하는 소스 컴포넌트의 사용 인터페이스와 서비스를 제공하는 타겟 컴포넌트의 제공된 인터페이스 사이에 존재하게 된다. (그림 3-a)는 두 프로덕트의 컴포넌트 뷰를 보여준다. 이들 컴포넌트 뷰는 앞의 (그림 2)의 휘처모델 예에서 어떤 휘처를 선택하느냐에 따라 서로 다른 아키텍처의 표현을 나타낸다. 그리고 하나의 휘처는 하나의 컴포넌트에 대응된다고 가정한다. 이 그림에서 두 프로덕트의 차이점은 컴포넌트 B와 컴포넌트 D의 위치이다. 이들 변화성을 지원하기 위해 아키텍처는 하나의 다이어그램으로 통합될 수 있다. (그림3-b)는 하나의 표현으로 두 개의 아키텍처를 표현한다[3].
또한 이들 Variant A의 가능한 구현 기능을 추적하기 위해 (그림 3-c)과 같이 나타낼 수 있다.

3.1 소프트웨어 컴포넌트의 변화성 모델링

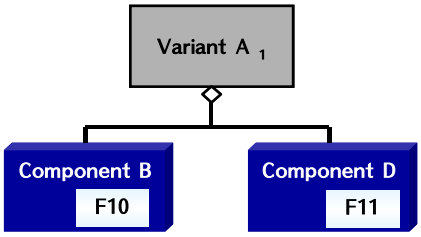
소프트웨어 프로덕트 라인에서 각기 다른 관점들에서 변화성이 어떻게 모델링 되는 가는 매우 중요하다. 유스케이스 모델에서의 변화점, 정적 모델에서의 추상 클래스들과 핫 스팟, 휘처모델에서의 휘처 모델링과 휘처 의존성을 통해서 다양한 모델의 관점에서 변화점을 나타낼 수 있다. 이런 소프트웨어 재사용의 개념은 다양한 관점 모델에서의 변화성을 가지고 다루어지기 위



(a) 두개 프로덕트를 위한 선택적인 아키텍처



(b) 아키텍처에서의 변화성



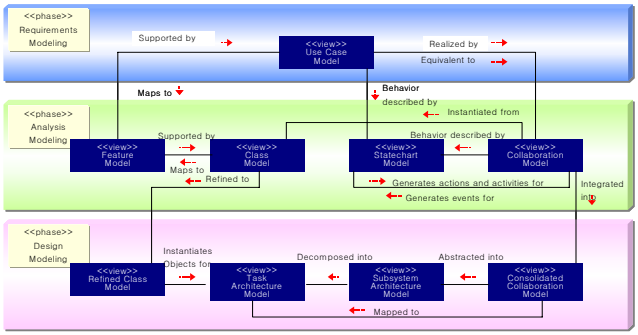
(c) 변화성의 선택

(그림 3) 프로덕트라인 아키텍처에서의 컴포넌트 변화성

해 사용되는 것이다. 또한 하나의 시스템들로부터의 선택적인 결정의 개념은 소프트웨어 프로덕트 라인의 협동 모델과 상태 모델에서의 변화성 설계를 위해 사용된다. 협동 모델과 상태 모델에서의 선택적인 분기와 메시지의 순서는 선택적이거나 변화 가능한 휘처가 선택되어졌을 때에만 사용이 가능하다. 또한 프로덕트 라인이 다양한 관점들에서 서로 모순이 없다는 것은 중요한 부분이다. 그래서 다양한 관점의 모델들 사이의 일관성을 유지할 수 있게 된다. 소프트웨어 프로덕트 라인의 다중 뷰 모델은 일반성과 변화성을 포함하는 다른 측면을 정의하는 객체지향 도메인 모델이다. UML을 사용한 여러 가지 모델들은 <표 1>에서와 같이 제시될 수 있다.

<표 1>UML을 사용한 여러 가지 모델

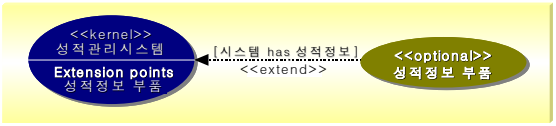
모 델	설 명
use case model view	유스케이스와 액터들에 관한 소프트웨어 프로덕트 라인의 기능적인 요구사항
static model view	관련성과 클래스들을 통한 소프트웨어 프로덕트 라인의 정적 구조
collaboration model view	· 소프트웨어 프로덕트 라인의 동적인 aspects · 객체들 사이를 통과한 메시지들의 순서
statechart model view	· collaboration model view와 함께 소프트웨어 프로덕트 라인의 동적인 aspects를 나타냄 · states와 states 사이의 transition을 정의
feature model view	feature/feature의존성, feature/class 의존성, feature/use case의존성, feature set의존성
class model view	· 시스템 내에 있는 클래스 간의 상호 작용 표시 · 객체들에 대한 청사진



(그림 4) 프로덕트라인의 다양한 뷰 사이의 관계

3.2 유스케이스 모델 뷰

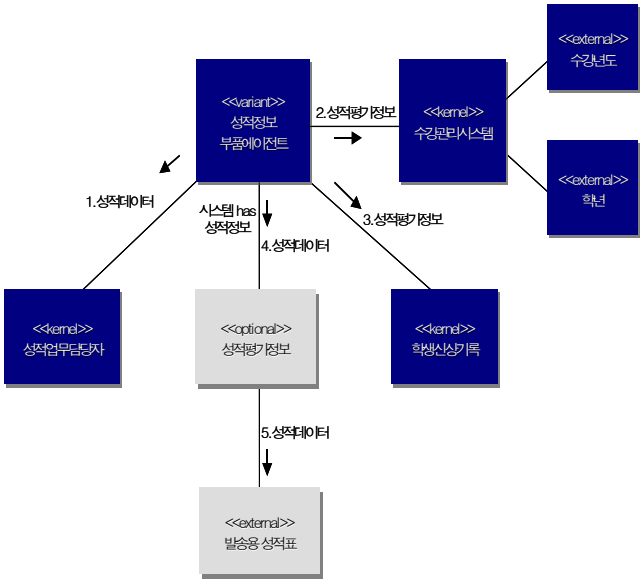
시스템의 기능적인 요구사항들은 유스케이스와 액터에 관하여 정의한다. 액터는 사용자 타입이고, 유스케이스는 블랙박스로 여겨지는 시스템과 액터 사이의 상호작용의 순서를 나타낸다. 프로덕트 라인의 일반성과 변화성을 나타내기 위하여 유스케이스는 kernel, optional variant 유스케이스로 분류된다. 유스케이스 모델에서의 전개는 유스케이스 변화점에서 표현될 수 있으며, 하나 혹은 그 이상의 위치에서 발생될 수 있다. (그림 5)는 성적 관리 시스템 프로덕트 라인으로써 성적정보 부품에서 시스템으로의 이동을 나타내는 유스케이스의 변화점을 제시하고 있다. kernel 유스케이스와 optional 유스케이스로 나타낼 수 있으며, 휘처 조건(시스템은 성적정보를 가짐)이 true라면, kernel 유스케이스로 확장되어지고, kernel 유스케이스의 "Extension points"로 이동한다.



(그림 5) 성적 관리 시스템의 변화점

3.3 협동 모델 뷰

협동 모델은 각각의 유스케이스와 연관된 객체들을 기술하고, 그것들 사이를 통과한 메시지들의 순서를 나타낸다. 유스케이스는 결정되어지고, kernel, optional, variant로써 분류되어지고, 협동 다이어그램으로 개발되어 질 수 있다. 프로덕트 라인 유스케이스를 가짐으로써 객체는 일반성과 변화성을 설명하기 위하여 kernel, optional, variant 객체로 분류되어질 수 있다. 애플리케이션 관점으로부터는 제어, 알고리즘, 엔티티, 인터페이스와 같은 규칙에 의존하여 객체가 분류되어 질 수 있다. 제어 객체를 실현시키는 유스케이스의 수집에 대한 전체적인 조정을 제공한다. 상태 의존 객체들 혹은 조정 객체들로써 분류되어진다. 알고리즘 객체는 응용 로직의 상세함을 포함한다. entity object는 데이터의 캡슐화를 제공한다. interface objects는 외부의 환경과의 인터페이스를 제공한다. 객체의 분류는 <<optional>>와 같이 UML의 스테레오 타입을 사용함으로써 설명되어질 수 있다. (그림 6)은 성적관리 유스케이스에 대한 협동 모델을 나타낸 것이다. 시스템이 성적정보를 가지지 않는다면, 기본 메시지 순서는 시스템이 "성적정보"를 가진다면, (그림 5)의 성적정보 부품에 상응하는 optional 유스케이스 "성적평가정보", "발송



(그림 6) 성적관리 시스템의 협동모델

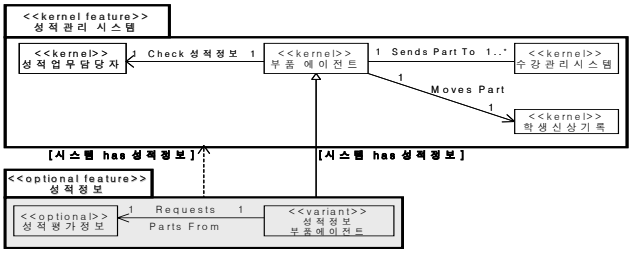
용 성적표"가 시스템이 추가되어진다. 만일 "성적정보"를 가진다면 휘처 조건은 식별하기 위해 사용되어진다. 만약 휘처 조건이 true라면, 즉 특정 프로덕트 라인 멤버가 이 휘처를 지원한다면 협동 다이어그램에서 분기하고, 그것은 휘처 조건에 의해 보호되어진다.

3.4 휘처모델 뷰

휘처는 최종 사용자의 기능적인 요구사항들을 말하는 것으로 그것은 소프트웨어 프로덕트 라인의 재사용 가능한 요구사항들을 명세하기 위해 사용되어 진다. 휘처는 소프트웨어 프로덕트 라인의 일반성과 변화성을 나타내기 위하여 kernel, optional, variant 휘처로 분류되어 진다. 후처 모델은 휘처 사이의 의존성을 통해 전개되어질 수 있다. 각 휘처는 하나 혹은 그 이상의 유스케이스에 의해 지원되어진다. 유스케이스 모델의 전개에서 대체적인 유스케이스는 특정한 조건하에 기본 유스케이스에서 확장될 수 있다. 유스케이스 의존성은 휘처 사이의 의존성에 대응하는 것이다. 또한 각 휘처는 하나 혹은 그 이상의 클래스들에 의해 지원되어진다. (그림 7)은 클래스 관련성에 기반한 휘처 의존성을 나타낸 것이다. 성적관리시스템 kernel 휘처는 성적업무담당자, 부품 에이전트, 수강관리시스템, 학생신상기록 클래스들에 의해 지원된다. 성적정보 optional 휘처는 성적평가정보, 성적정보 부품 에이전트 클래스들에 의해 제공되는 것이다. 휘처들 간의 휘처 의존성은 성적정보 부품 에이전트와 요소 에이전트 클래스들 사이의 상속 의존성에 반영된 것이다.

3.5 소프트웨어 프로덕트라인의 메타 모델링 방법

다양한 관점들 사이의 관계는 UML 표기법을 사용한 메타 모델에서 그들 스스로 모델링 되어질 수 있다. 다중 뷰 메타 모델은 어떻게 뷰를 다른 뷰로 연관시킬까 하는 것을 나타낸다. 메타 모델은 메타 클래스들과 그들의 속성 그리고 그 사이의 관계를 정의한다.



(그림 7) 클래스 관련성에 기반한 휘처 의존성

소프트웨어 프로덕트 라인 개발 단계는 각각의 뷰에 포함될 수 있고, 각각의 뷰는 메타 클래스들로 분해된다. 사용자 정의 다중 뷰 모델은 메타 모델의 인스턴스이다. (그림 4)의 요구사항 단계에서의 유스케이스 모델은 액터와 케이스에 관한 다양한 관점 모델의 기능적인 요구사항들을 나타내고, 분석 단계에서의 휘처모델은 휘처들과 그것 사이의 의존성이 의미들에 대해서 소프트웨어 프로덕트 라인의 일반성과 변화성을 나타내고, 클래스 모델은 클래스를 통한 다양한 관점의 모델의 정적인 구조를 나타내고 협동 모델은 객체들과 그것들의 대화 메시지들을 기술하는 것에 따라서 다양한 관점 모델의 동적인 양상을 나타낸다.

4. 결론

본 논문에서는 재사용 가능한 아키텍처를 설계하기 위해 변화성의 명확한 표현과 컴포넌트의 변화성을 설계하기 위해 다중 뷰의 모델링을 통하여 프로덕트 라인의 다양한 측면을 제시하였다. 이 방법은 다중 뷰 메타 모델을 사용하는 각기 다른 뷰 사이의 관계를 정의함으로써 다양한 뷰들을 통합시킨다. 메타 모델은 생명주기 단계와 각 단계에서의 관점들, 그리고 각 관점들 안에서의 메타 클래스들로 설명된다. 다양한 관점 모델링 접근 방법의 장점은 각 관점들의 변화점을 정확히 모델링하고, 변화점 사이의 관계들을 정의함으로써 소프트웨어 프로덕트 라인의 전개를 가능하게 하는 것이다.

참고문헌

[1] Hassan Gomaa, "Multiple-view Meta-Modeling of Software Product Lines", IEEE on ICECCS, 2002
[2] Roger S. Pressman "Software Engineering A Practltiners' Approach" 3rd Ed. McGraw Hill
[3] 김행곤 외, "프로덕트 라인 메타모델 정의와 변화성 모델링", 한국정보처리학회 춘계학술발표회지, 제 10권 제1호, pp 1709-1712, 2003.
[4] Maarit Harsu, "A Survey of Product-Line Architectures", Software Systems Laboratory Tampere University of Technology, 2001.
[5] Felix Bachmann, "Managing Variability in Software Architecture", SSR;01, pp126-132, 2001
[6] Matthias Riebisch, "Extending Feature diagrams with UML Multiplicities", IDPT, 2002
[7] Detleft Streitferdt, "Extending the UML to model system Families", IDPT, 2000