Work Products for Integrated Software Development

Bernhard Deifel, Wolfgang Schwerin, Sascha Vogel

TUM-I9921
Dezember 99

Work Products for Integrated Software Development[1]

Bernhard Deifel, Wolfgang Schwerin, Sascha Vogel

Institut für Informatik
Technische Universität München
Arcisstraße 21
80290 München, Germany
(deifel|schwerin|vogels)@in.tum.de

Abstract:

Integration of development processes for different kinds of systems, such as information and embedded systems is an important topic when we build software systems for application domains that are strongly integrated. Besides that, different project views, such as development and management must also be well integrated in process models. In this paper we propose a work product oriented basis for integrated process models. We define a model of work product, covering and integrating the fields of requirements engineering, architectural design and project management.

***Keywords**: Process Modeling, Product Model, Requirements Engineering, Design, Project Management*

[1] This work is supported by the Bayerische Forschungsstiftung within the FORSOFT research consortium

1 Introduction

With the increasing complexity of software systems and hence of development processes, modeling of software development processes becomes more and more important and underlies evolution similar to the evolution of notations. Regarding for example the UML [RJB99] and its standardization by the OMG, we recognize that strong efforts are made concerning the integration of different complementing notations. In the field of development processes integration is an equally important issue. In general and especially in the interdisciplinary FORSOFT research cooperative, which consists not only of computer scientists and researchers from mechanical and electrical engineering, but also of economy experts and practitioners from leading companies in each of these four fields, we observe that software systems are increasingly applied in application domains which strongly relate different domain spaces. For example, nowadays enterprise logistics and accountancy are closely coupled with software based control of production machines (PCL controls). Due to this heterogeneous character of software systems, we have to integrate specific development process models like the ones for business software and of embedded systems.

Equally important is the integration of different parts of development processes, for example the different management tasks such as project and configuration management, with development tasks, such as requirements engineering and design.

Integration usually increases a model's complexity. In order to manage the complexity of integrated process models we separate the description of work products, that is all intermediate and end results of the development process, from the description of development activities and strategies. This separation has for example been applied in [BLR+95, IABG97, LT96, JBR98, Kru98]. There, relationships and dependencies between work products are only covered implicitly in the description of development activities. Compared to this, in further approaches [MBJ+90, Jeus92, RRP99] relationships and dependencies between work products are modeled explicitly as elements of a model for work products.

Models of work products which state product relationships and dependencies explicitly, increase comprehensibility by providing us an integrated view of development processes without having to consider both, work products and development activities. Based on this integrated work product view we then can define development activities and strategies, which respect relationships and dependencies between work products.

Relationships and dependencies between work products are also important for process integration, because they allow us to relate work products from different development tasks and application domains explicitly. For example, we can relate "results" from project management with "requirements" and "architectural elements" from requirements engineering and design, respectively.

In this paper, we propose a model of work products which provides the basis for the integration of development processes for different application areas and for the integration of different development tasks. Thereby we consider topical questions, such as the relationship between functional and non-functional requirements with design decisions, and the association of management views with requirements and design views. This means, that we focus on requirements engineering, architectural design, and project management.

Regarding requirements engineering and architectural design, we decided to apply a decision-oriented approach, covering negotiation and selection of requirements and architectural alternatives. Thereby we related

- ideas of refinement hierarchies from goal-based requirements engineering [LW98, MCY99] with a decision-oriented process view known for example from [RG94] and a model for capturing agreement knowledge as presented in [Pohl96],

- architectural styles [SG96] with decision-oriented design, and

- project management not only with tasks and system architecture but also with requirements.

We define the product model detailed and in a schematic way, so that it provides directly a structure for project participants in which they can write down and collect information. Therefore, the product model can serve for example as a metamodel for CASE tools.

We structure the rest of this paper as follows:

In Section 2 we introduce the building blocks of a product model in general. Then, in Section 3 we describe the principle of decision-orientation, which we apply in the product models for requirements engineering and design. Within the subsequent three sections we explain aspects and work products of requirements engineering, design, and project management, respectively. Afterwards we integrate the three tasks and focus on comprehensive dependencies between their work products. The last section concludes our work and gives an outlook on future work.

2 Product Model Elements

Before we introduce the product models for the three aforementioned tasks, we explain how we characterize product models.

In general, a product model consists of a set of product types and relationship types characterizing relations between products and product types. With respect to relationships between products, we distinguish association, aggregation, and dependency. Additionally, we relate product types by a generalization/specialization relationship, so that we structure the set of product types hierarchically. The definition of a product type covers its attributes, its parts and its dependencies. Each product type can be identified by its unique name and shorthand. For each product type we provide its purpose and description.

In order to represent and structure the information covered by a product, each product type defines attributes. We model “elementary” product properties by untyped attributes. A certain product (instance of a product type) and its state during the development process are then determined by the actual attribute values. An example of an attribute is the “status” of a requirement, which is either “under_negotiation” or “agreed”. A constraint for the “contract” product could then demand that only those requirements may be part of the contract whose status is agreed.

As already mentioned above, we define the following relationships between products:

- association,
- aggregation, and
- dependency.

We use associations in the sense of relationships in Entity/Relationship models. Hence, associations represent a certain information on their own. We can ask questions such as which products play a certain role within a certain association. For simplicity, we use only binary associations. Furthermore, we restrict ourselves to undirected associations because of directed associations are often related with the notion of navigability, which is not relevant in the context of our product model. An example of an association is the relationship between stakeholder and requirement. This association documents information about which requirement is required by which stakeholders.

In addition to association, we use aggregation. With aggregation we apply the principles of grouping and hierarchical structuring of products. An aggregating product, which we will also call a “container” in the following, can serve us for introducing a term that stands for the group of aggregated products, which we will also call “parts” in the following. For example, we introduce the term “domain model” standing for “domain data view” and “domain process view”. Note however that we assume aggregation not to express mere grouping, but hierarchical structuring of products on an instance level. Therefore aggregation implies a partial order on the instance level. This is an important difference from association. In contrast to other approaches, we do not relate lifetime dependency, visibility restrictions or exclusivity with aggregation, because of these aspects being not of primary concern for our product model.

Besides aggregation and association, we define dependency relationships between products. A dependency relationship is always directed, either uni- or bidirectionally. A dependency relationship expresses the fact, that the state of a product, that is its attribute values, is dependent on the state of other products. This means, that when a product is dependent on another product, then changes of the latter can imply changes of the former. We use informal text for the description of *how* attribute values of products are (functionally) related. For example, we could state that the priority of a requirement has to be the average of the priorities of the requiring stakeholders. Note that, whereas instantiations of aggregations and associations generally vary over time and between product model instances, by means of dependencies we define invariant properties which must hold for all product model instances.

Finally, we define a generalization/specialization hierarchy on product types. We use it in the sense of an „is-kind-of“ relationship which implies, that a product type being the specialization of other product types, "inherits" the attributes from the more general types as well as the roles the more general types play in relationships.

By means of generalization relationships between work product types, we have the possibility to define and relate generic work products, which we can refine to domain specific elements. Thereby, integration on the generic level of abstraction is relevant for all domain specific specializations. For example, management is mainly based on an abstract nontechnical view of a system's architecture, consisting of hierarchically structured architectural elements. Whether these elements are refined to objects in an object-oriented sense or to pipes and filters (cf. [SG96]) is not important from the management point of view. Therefore, for an integrated view on management and architectural design we relate work products for management, such as project results, with these generic, hierarchically structured architectural elements. Thereby integration of management and architecture remains valid for specific specializations of architectural elements.

For graphical illustration of product types and their relationships, we refer to the syntax of UML class diagrams, exemplified in Figure 1.

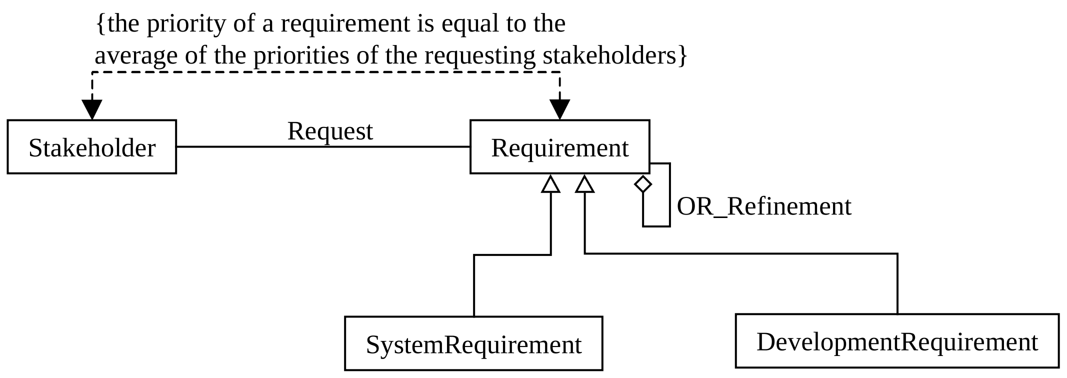
{the priority of a requirement is equal to the average of the priorities of the requesting stakeholders}
Stakeholder
Request
Requirement
OR_Refinement
SystemRequirement
DevelopmentRequirement

Figure 1: Exemplary class diagram defining product types and relationships.

Besides graphical class diagrams we use schemata in a tabular style for detailed description of the elements of a product model. These schemata are given below.

Product type	The name of the product type.
Description	Explanation of the information covered by instances of this product type.
Purpose	Explanation of the methodological function and significance of the product type.
Attributes	Explanation of "elementary" product properties.
Aggregations	Explanation of the aggregations in which instances of this type play the aggregating role.
Dependencies	Explanation if and how attribute values of instances of other product types depend on attribute values of this type's instances.

Scheme for product types, including sections for aggregations and dependencies

Association type	The name of the association type.
Description	Explanation of the information covered by an association of this type.
Purpose	Explanation of the methodological function and significance of the association type.
Attributes	Explanation of "elementary" association properties.

Scheme for association types

In order to support document and change management, and verion control, we define that any product and association type has the following attributes:

- productname, i.e. the name of an instance, e.g. System1_RequirementsSpecification
- id, that is a unique instance identifier
- creator, i.e. the project participant who created the product
- date_of_creation, i.e. when the product was created
- change_log, i.e. which changes have been applied on the product
- version_nr, i.e. identifier of the actual version
- status, exemplary values are "existent" and "under_negotiation".

3 Decision-Orientation

When developing systems, we continuously have to take decisions. Regarding requirements engineering, we have to decide which requirements of the set of elicited requirements are to be realized. Besides that, when designing a system's architecture, we have to take decisions in the sense of selection of adequate architectural alternatives. A decision-oriented view on the development process leads us to work products which support taking and documentation of decisions. Note that this view on the development process does not imply a timely order in the sense, that *all* decisions have to be taken before succeeding with development steps in which we produce results being related with the decisions. For example, requirements need not necessarily be selected before we start with architectural design. The work products that we derived from the decision-oriented process view merely provide us with information structures to document decisions, independently from when these decisions are taken in process enactment.

Generally decision-oriented approaches consist of the following basic elements:

Alternatives: For a given problem space different possible solutions are defined.

Decision criteria: The problem space has to be evaluated with respect to different criteria which provide objectives for the selection of one of these alternatives. Usually the alternatives fulfill the criteria in different levels. In order to be able to compare the alternatives with respect to the fulfillment of a criterion some kind of metrics for each criterion has to be defined. Alternatively the different alternatives can be compared mutually with respect to the fulfillment of a criterion.

Classification with respect to decision criteria: Each alternative is classified with respect to the fulfillment of the different criteria.

Rationale: The rationale for the selection of an optimal solution is defined. For this purpose usually the decision criteria are weighted mutually.

Solution: From the given alternatives the best one with respect to the given rationale is selected. This represents the decision solution.

In Sections 5 and 7 we assign concrete work product types to the basic elements mentioned above.

4 Requirements Engineering

In requirements engineering we try to achieve a correct understanding of stakeholder needs, so that we can specify a validated, consistent and complete set of requirements (cf. [Pohl96, Dei98b]). Requirements refer to the software system that has to be developed or modified, and to the development process.

Usually different stakeholders with different viewpoints and different needs are involved in a project. Moreover, we have to consider a variety of development methods, programming techniques, software platforms, available frameworks and components etc. in a development process. This leads us to the elicitation of alternative requirements which have to be documented.

Multiple and possibly conflicting viewpoints as well as conflicts between needs and constraints force us to strive for an acceptable rather than a maximal or optimal solution. This means, that we have to negotiate, prioritize and select (alternative) requirements. Therefore, besides the documentation of alternatives, the documentation and justification of decisions is an important task. Thereby we can enhance traceability of requirements which in turn leads to a better understanding of intentions and supports change management, for example. It is here, where decision-orientation comes into play.

According to the fact that we usually elicit, negotiate, change and specify requirements at different stages of a development process, we consider requirements at different levels of detail and more or less structured. When we relate these different levels we get a hierarchical structuring of requirements. By that means we improve comprehensibility and traceability.

In a project, we must not only determine "what" we have to develop, but also the way "how" we develop it. Consequently, we must consider requirements for the system to be developed, as well as requirements for the development process. System requirements state required functional and nonfunctional system properties. Development requirements concern properties of work products and development activities. Often requirements of these two kinds are closely related. For example, high safety requirements may require usage of formal methods and notations.

A software system is supposed to play a certain role in an application domain, by providing certain domain functionality. Examples for application domains are banking or car control. Explicit modeling and documentation of domain knowledge helps us to get a good understanding of a system's domain context and to describe domain specific requirements adequately. Besides that, we can thereby describe how a system is embedded in its environment.

5 Model Elements for Requirements Engineering

With the product structure given below, we provide a basis to cover the aforementioned information. The product structure enables us not only to document the final results of requirements engineering in one document, usually called a requirements specification, but we document also intermediate results and taken decisions.

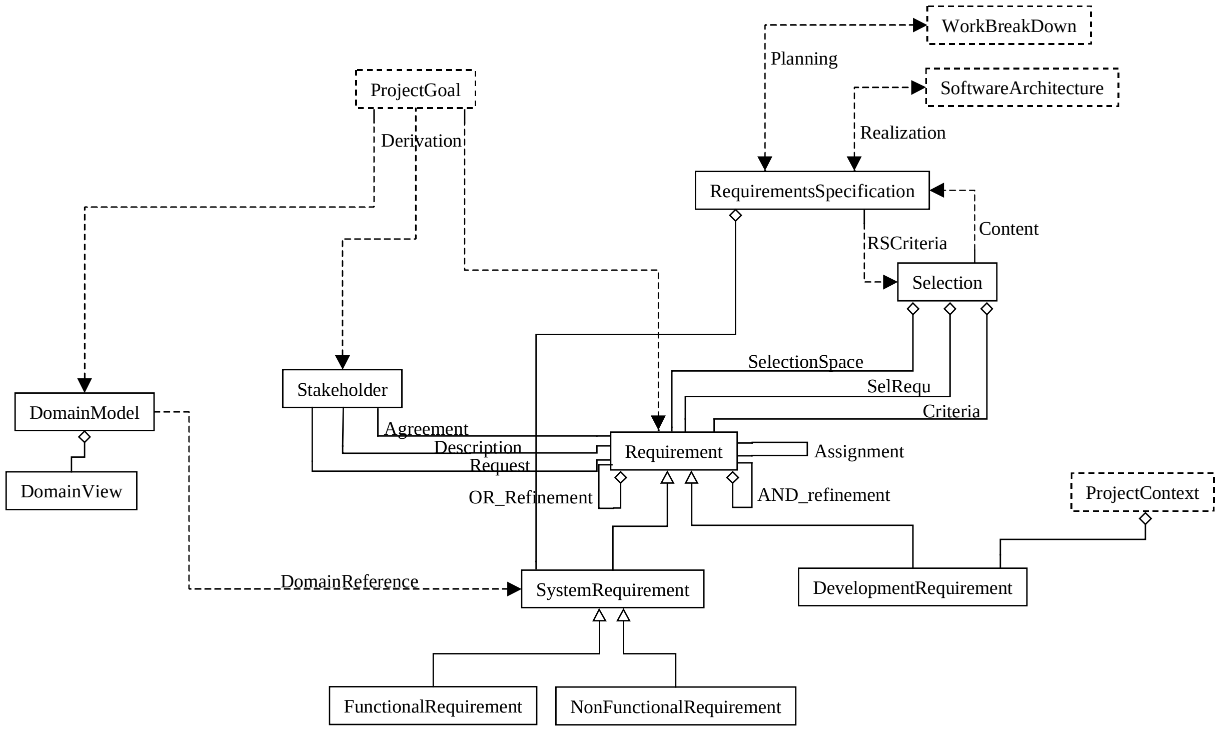

Figure 2: Product structure for requirements engineering

In Figure 2 we describe the product structure for requirements engineering graphically. The product types *WorkBreakDown*, *ProjectGoal* and *ProjectContext* are explained in detail in Section 9, product type *System Architecture* is explained in Section 5. In Section 10 we treat the dependency relationships *Realization* and *Planning*.

For requirements engineering we assume work products of type *ProjectGoals* and *ProjectContext* (cf. Section 9) to be given. Based on this information we identify relevant stakeholders, gather knowledge about related application domains and elicit requirements for the envisioned system and development.

5.1 Stakeholder

We elicit domain knowledge and requirements from identified stakeholders. Stakeholders have needs and constraints, that is they request requirements. Apart from requesting requirements they describe requirements or certain aspects of them. For example, a potential system user can describe user-system interaction for a functional requirement, which is requested by another stakeholder. Stakeholders can agree or not with a certain requirement. We must hold this information for negotiation and decision purposes.

Product type	Stakeholder
Description	A stakeholder stands for a person or organization or a group of persons/organizations, who have some kind of justified interest in the system to be developed. Stakeholders can request, describe and agree with requirements.
Purpose	Associating stakeholders with requirements (agreement, description, request) is a prerequisite for negotiation and validation of requirements.
Attributes	Importance: informs us about how important it is to respect the stakeholder's point of view, for example when we select an alternative.
Aggregations	-
Dependencies	-

5.2 Requirement and Refinement

A requirement expresses a certain quality or capability of a system or a development process. We need the concept *Requirement* because requirements are the units of information which we define, refine, negotiate and select in requirements engineering. Requirements engineering methods are based on this concept.

We classify requirements accordingly to what they determine (cf. [KS98, LK95]):

- System requirements determine what functionality a system has to provide (functional requirements) and qualities of how this functionality is to be provided (nonfunctional requirements).

- Development requirements do not determine what we develop but how we develop it, that is the development process.

We structure requirements hierarchically by means of OR- and AND-refinement aggregations (cf. [LW98, MCY99]). This allows us to express explicitly

- alternatives and what they have in common (OR-refinement),

- how different requirements contribute to a more general requirement (AND-refinement),

- requirements at different levels of detail (AND-refinement).

OR-refinements represent points of negotiation and decision.

AND-refinements enhance comprehensibility and traceability by relating requirements at different levels of detail and showing the parts of which composite requirements are made up.

Product type	Requirement
Description	During development we state requirements in a more or less detailed and structured way. Especially in early stages of development, a requirement may cover and relate several kinds of requirements, e.g. requirements concerning the envisioned system as well as the development process. Example: Develop a safe and secure break control system and use the UML [RJB99] for the design specification. The product type *Requirement* allows us to cover this integral kind of information which can be related with more detailed and structured representations (see AND- and OR-Refinement aggregations).
Purpose	A requirement can be the impetus for further refinements. For traceability reasons we must be able to document such initial information. By means of the OR- and AND-Refinement aggregation, a *Requirement* product helps us to structure requirements. We can base the negotiation task on the aggregation of alternatives (OR-Refinement). The AND-Refinement allows us to state explicitly to which integral requirements certain sub-requirements contribute. Thereby we can support traceability and multiview-oriented requirement description. We can base consistency analysis on this aggregation.
Attributes	DescriptionStyle: Classification dimensions for the description style are: - operational vs. declarative - exemplary vs. complete (e.g. a scenario can be interpreted as a set of *some* or of *all* possible actor interactions) - positive vs. negative (e.g. scenario which has to be supported vs. one which has to be excluded) - formal vs. informal (with respect to syntax and semantics). Stability: Probability of change. Relevant e.g. for the design with respect to flexibility. Priority: Importance, which can be relevant for or result of negotiation.

Aggregations	AND_Refinement: Aggregation of all requirements that contribute to satisfy the aggregating requirement. This means, that we have to satisfy all sub-requirements to satisfy the whole. Aggregated requirements cover only some aspects of the aggregating requirement. OR_Refinement: Aggregation of alternative refinements. This means, that satisfying one of the refined requirements is sufficient for the satisfaction of the superposed requirement. Nevertheless, multiple alternatives can be chosen to be realized. We may then distribute alternatives over different versions of a system or provide possibilities for configuration. If a requirement is refined, then it is either OR refined or AND refined, but not both at a time. We do not define any constraints on refinement with respect to the classification hierarchy of requirements. However in some cases this might be justified, e.g. a functional requirement must not be refined by a nonfunctional one.
Dependencies	-

Product type	SystemRequirement
Description	A system requirement is a condition or capability that must be met or possessed by a system to satisfy stakeholder needs, a contract, a standard or a specification. (cf. IEEE standard IEEE-610.12, 1991). Similar to a general requirement, a system requirement may cover and relate several kinds of requirements, e.g. functional as well as nonfunctional ones. Example: The system requirement, that a system has to support data transfers between remote branches in a secure and efficient way, contains functional as well as nonfunctional information.
Purpose	This product type allows us to document requirements on a system, which comprise functional as well as nonfunctional aspects.
Attributes	-
Aggregations	-
Dependencies	-

Product type	FunctionalRequirement
Description	A functional requirement describes a service, that is required by some stakeholders to be provided by the system. Example: The system has to support data transfer between branches. A functional requirement usually comprises information about which parts of the domain world have to be represented in a system, which domain processes must be supported by the system, and how user-system interaction has to look like (including information about work sharing between system and user).
Purpose	Functional requirements determine the purpose of a system, that is “what” a system has to do, and the relationship between the domain world and the system.
Attributes	-
Aggregations	-
Dependencies	-

Product type	NonFunctionalRequirement
Description	A nonfunctional requirement represents stakeholder needs, concerning the system, which are qualities of the system in general or of certain other requirements. Examples: The system has to be secure and safe. The "look and feel" of GUI elements must meet a certain standard.
Purpose	A nonfunctional requirement guides or constrains the architecture or the implementation of a system. Thereby, it has influence on “how” system functionality is realized.

Attributes	DescriptionStyle: A design constraint is either formulated in a constructive or declarative style. Constructive style means that a requirement defines a part of a system architecture or a system implementation, respectively. Constructive requirements have therefore to be integrated (by composition) in a system architecture or implementation. Example: Use a certain GUI toolkit (in order to provide a standard compliant look and feel and to achieve maximal reuse within an enterprise). Declarative requirements define nonfunctional qualities of a system without already defining part of an architecture or implementation. Declarative requirements have therefore not to be integrated in an architecture or implementation, but to be respected when taking design decisions. Examples: “Provide a standard compliant GUI look and feel”. "Strive for high runtime efficiency." The former may influence the choice of a GUI toolkit, whereas the latter can be relevant for selecting an appropriate sorting algorithm, for example. Classification: Enumeration-type. Describes the different possible peculiarities for a given nonfunctional requirement and their meaning. The classification of a nonfunctional requirement is important for the evaluation of architectural alternatives and architectural styles.
Aggregations	-
Dependencies	-

Product type	DevelopmentRequirement
Description	A development requirement represents a guideline or constraint on “how” a system is to be developed or described. A development requirement refers either to development activities or work products. Examples: Documentation guidelines and standards, development techniques, engineering principles, time and cost constraints. Example for a requirement on a work product: In order to ensure properties of a requirements specification concerning the representation of requirements, stakeholder agreement on requirements, and completeness of the description of requirements, we could for example define: Functional requirements contained in the requirements specification have to be formulated by means of UML activity, class and sequence diagrams (representation). All stakeholders, which have requested a certain requirement must agree with chosen alternatives (agreement). All agreeing stakeholders that are also users must have defined an according use case (completeness). Example for a requirement on a development activity: For coding, use a certain CASE-Tool and its code-generation facilities as far as possible.
Purpose	Development requirements determine characteristics of development activities and work products. For example, assignment of development techniques, strategies and notations to activities and work products (cf. method allocation of the V-Modell [IABG97]) can be expressed in terms of development requirements.
Attributes	-
Aggregations	-
Dependencies	-

5.3 DomainModel

Besides and in combination with requirements we also elicit knowledge about a system’s application domain from stakeholders. We abstract from unimportant details and model relevant domain knowledge explicitly. Usually we will describe a domain model from different points of view, for example a structure-oriented and a process- and information-flow-oriented view. Languages, such as the UML provide possible notations for the representation of these views.

We cover the fact, that requirements are related with elements of a domain model by a dependency relationship. For example. when a requirement expresses that a certain domain process is to be supported by the system, then this requirement is dependent on the according domain process, on user-system-interaction descriptions for this process and on domain structures being relevant for this process.

System requirements may not be requested directly by stakeholders, but derived from analysis of the according application domain. For example, when a stakeholder wishes a certain domain process to be supported by a software system, this constitutes a functional requirement. A domain model may contain time limits for the execution duration of this process, for example due to physical or contractual demands. From this domain knowledge we then derive an appropriate performance requirement. This means, that on the one hand we derive functional as well as nonfunctional system requirements from a domain model. On the other hand we express system requirements in terms of a domain model. In any way, system requirements are dependent on domain knowledge.

Product type	Domain Model
Description	A domain model is an abstraction of the real world covering domain knowledge that is relevant for system development. In a domain model we state knowledge of an application domain explicitly. When we assign domain elements to a system, e.g. certain domain processes that are to be supported by a system, we cover information about the environment-system border, too. We model the domain world independently from aspects concerning their technical implementation in a system.
Purpose	Explicit modeling of an application domain increases understanding for the domain context of a system. Moreover, a domain model serves as a reference for the representation of domain elements, such as business processes and domain data, in a software system. Besides that, a domain model describes the work-partitioning between a system and its environment (environment-system border).
Attributes	-
Aggregations	A domain model serves as a container for different views on an application domain. Examples for such views are static structure and process view.
Dependencies	Dependency “DomainReference”: A *DomainReference* dependency describes which domain elements are referenced in a given requirement and therefore determine a requirement’s properties. For example, when a domain process changes then the requirements for a system which supports this process change accordingly. A *DomainReference* dependency makes the relationship between an application domain and a system explicit. It serves as an explanation and justification of required system properties. We can thereby trace changes in the application domain to system properties.

Product type	Domain View
Description	A domain view covers certain aspects of a domain model.
Purpose	By means of domain views we can describe different aspects of a domain model separately. Thereby we can divide a complex whole, the model, into less complex parts, the views. This helps us to increase comprehensibility of the whole domain model.
Attributes	-
Aggregations	-
Dependencies	-

In the following we give an example of how a domain model could be composed. The composition of a domain model is usually dependent on the according application domain. We assume the three views, illustrated in Figure 3 , to be suitable for information systems. They may have to be altered and/or other views may have to be added when we want to model reactive, embedded and real-time systems.

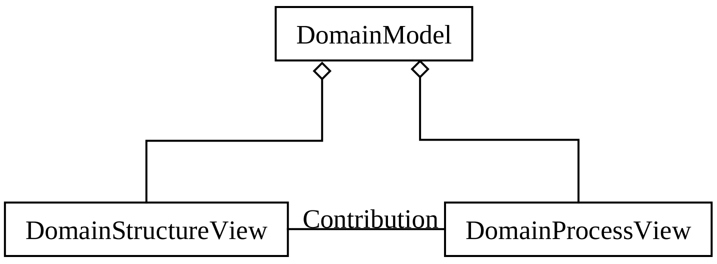

Figure 3: Exemplary composition of a domain model

Product type	DomainStructureView
Description	In a domain structure view we describe the static structure of an application domain's elements, and their static relationships. We do not cover behavioral aspects. By elements we think of real world objects, e.g. a coin, or real world concepts (virtual objects), e.g. a bank account, or actors, e.g. a bank clerk or an ATM (automated teller machine). Examples for static aspects are state space descriptions of elements, and compositional (whole-part) as well as association relationships between elements. A structure view may be represented by class diagrams (cf. [JBR98]).
Purpose	By abstraction from behavioral aspects, a domain structure view provides us an overview about terms of an application domain, their static structure and relationships. Prerequisites for behavioral descriptions are defined here. For example, the types of objects, which are changed when processes are enacted, are defined in a structure view.
Attributes	-
Aggregations	-
Dependencies	-

Product type	DomainProcessView
Description	A domain process view is a process-oriented view on the domain world. Enactment of a process changes states of domain objects. Processes are enacted by optionally assigned actors. An actor is a person or a system taking an “active” part in the enactment of a domain process, e.g. a bank customer in the process of a money transfer. The system to be developed is a specific actor. Within a process model we also define the flow of objects between processes, which introduces a (causal) order on processes. Processes can be hierarchically decomposed. Variants of domain process views differ with respect to further assumptions on process properties, such as isolation and atomicity. Widely used process views for business domains can for example be found in [VB96]. A process view may for example be represented by petri-nets or activity diagrams (cf. [JBR98]).
Purpose	A domain process view provides information about domain functionality. When functional requirements require the support of a domain process by a system, the according domain process view has to be respected. The system has to represent related domain objects, and establish object flow with related actors by means of adequate system-interfaces. With assigning a system as an actor to certain domain processes, we define one aspect of the system-environment border.
Attributes	-
Aggregations	-
Dependencies	-

Association type	Contribution
Description	A *Contribution* association states which domain elements contribute to a domain process, either as actors or input/output elements.
Purpose	This kind of association contributes to the integration of structure and process view.
Attributes	-

5.4 Assignment, Selection and RequirementsSpecification

In order to express constraints on requirements, we assign requirements to other requirements. Thereby we can describe for example which nonfunctional requirements are relevant for which functional requirements.

The assignment relationship is particularly important for the selection from alternatives. Requirements assigned to the root of alternatives represent qualities requested from all alternatives.

Therefore these assigned requirements represent the criteria on which we base our selection of the most suitable alternatives. We prioritize these criteria if necessary and evaluate all alternatives mutually and with respect to them. For each selection we request a rationale, which explains and justifies this evaluation. In Table 1 we assign model elements for requirements engineering to the basic elements of decision-orientation (cf. Section 3).

Decision Element	Product Model Element
Alternatives	*OR-Refinement* aggregation of work product type *Requirement*
Decision criteria	Association type *Criteria* between work product type *Selection* and *Requirement*
Classification	contained in Attribute *Rationale* of work product type *Selection*
Rationale	Attribute *Rationale* of work product type *Selection*
Solution	Association *SelRequ* between *Selection* and *Requirement*

Table 1: Decision elements and related model elements for requirements engineering

A requirements specification defines those requirements that have to be met by a system. Usually we demand certain properties of a requirements specification, e.g. that functional requirements are formulated in a certain style, and a certain degree of stakeholder agreement on the contained requirements. These properties are themselves requirements, namely development requirements assigned to a requirements specification product. We understand these properties as part of the criteria for the selection of alternatives.

Association type	Assignment
Description	Requirements can express required qualities of other requirements. An *Assignment* association expresses which requirements state qualities (role *Quality*) of which other requirements (role *Objective*). Example: For some functional requirements high runtime efficiency is a requested quality, whereas for others it is storage efficiency.
Purpose	An *Assignment* association covers important traceability information, supporting change management. By means of an *Assignment* association, we can express relationships between different kinds of requirements, e.g. the assignment of nonfunctional requirements to functional ones.
Attributes	-

Product type	Selection
Description	A selection describes which alternatives (OR-refinements) of a given requirement (SelectionSpace) are most suitable with respect to certain criteria. A rationale explains and justifies a selection. A selection may comprise not only one but several alternatives. This is admissible and sensible especially in combination with the distribution of alternatives over different variants or versions of a system.
Purpose	Selections document decisions that we take. They thereby enable traceability.
Attributes	Rationale: Explains and justifies the evaluation of SelectionSpace elements. All regarded alternatives are evaluated with respect to the same criteria, and relative to each other.
Aggregations	SelectionSpace: Requirement whose alternatives (OR-refinements) we consider. Criteria: Covers those requirements, according to which all elements of a SelectionSpace are mutually evaluated. Criteria can be prioritized. SelRequ: Covers those elements of a SelectionSpace, that are evaluated to be the most suitable with respect to Criteria.
Dependencies	The criteria on which we base our selection (Criteria) are the requirements that are assigned to the requirement (SelectionSpace) from whose alternatives we determine the selection, as well as development requirements assigned to a requirements specification. Dependency "Content": Only selected requirements can be elements of a requirements specification.

Product type	RequirementsSpecification
Description	A requirements specification contains all and only those system requirements, that have to be met by an envisioned system. This means, that all and only those system requirements are contained, which are either choices of alternatives (OR-Refinement), or AND-refinements of chosen requirements, or requirements assigned to one of the former kind. Thereby, a requirements specification represents the result of negotiation and selection of requirements. There exists only one requirements specification per project. Yet, different versions of this product can exist.
Purpose	As part of a contract, a requirements specification lays down requirements that have to be met by a system. As part of a documentation, a requirements specification contains the requirements met by a system, from stakeholders point of view. Consequently, the evaluation and selection of architectural alternatives, as well as the structuring of parts of a work break down are related with the constituent requirements of a requirements specification.
Attributes	-
Aggregations	-
Dependencies	Dependency “RSCriteria”: Development requirements assigned to a requirements specification define criteria for selections.

In the example illustrated in Figure 4 and represented as an object diagram (cf. [RJB99]), we show a product structure documenting the selection of a requirement from alternatives. In this example the functional requirement, that the envisioned system has to be configurable is refined by the requirements, that an according configuration logic and configuration interface have to be provided. Stakeholders are administrator and end user, who both request different qualities of the configuration interface. The administrator wants remote configuration based on scripts to be supported, whereas the end user puts emphasis on an intuitive and guided usage. Two alternatives for the configuration interface are a graphical or an ASCII-based interface. Both would not meet the requirements of administrator and end user. We defined a third alternative, requiring that the system must provide a graphical interface per default and a ASCII-based interface on demand. This alternative is selected because of meeting both the administrators as well as the end users demands.

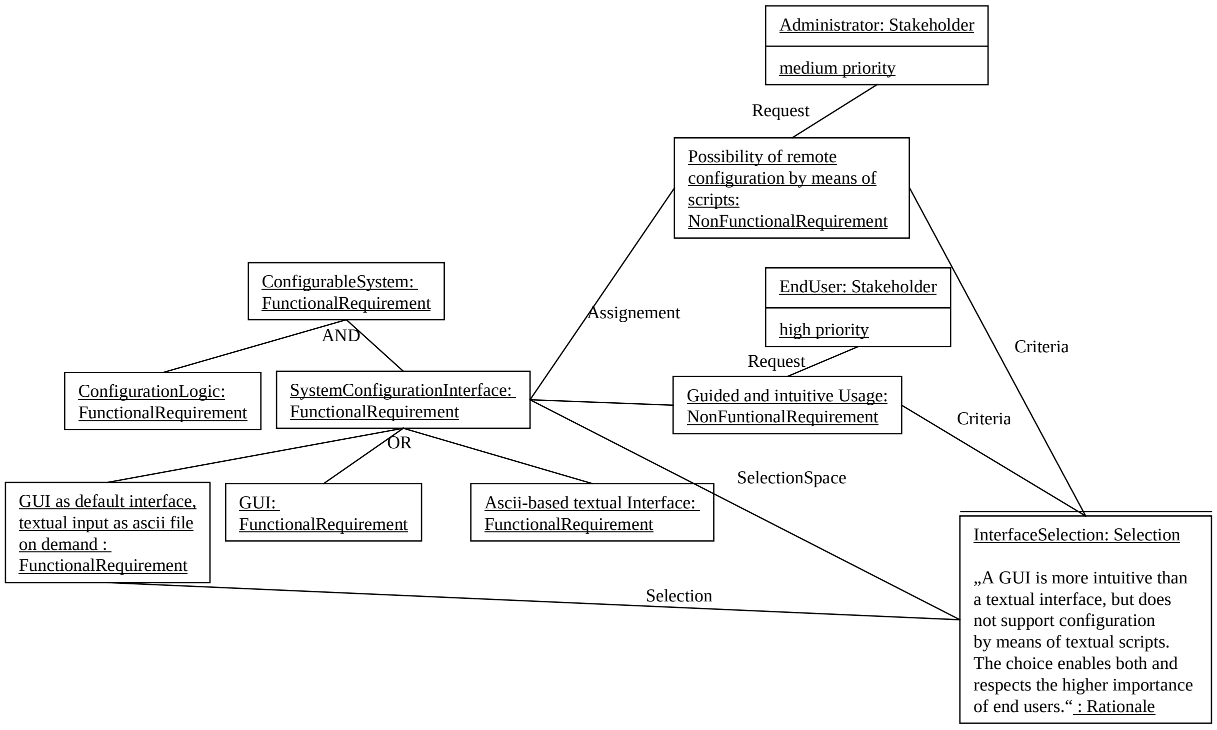
Administrator: Stakeholder
medium priority
Request
Possibility of remote configuration by means of scripts: NonFunctionalRequirement
ConfigurableSystem: FunctionalRequirement
AND
Assignement
EndUser: Stakeholder
high priority
Request
Criteria
ConfigurationLogic: FunctionalRequirement
SystemConfigurationInterface: FunctionalRequirement
Guided and intuitive Usage: NonFuntionalRequirement
Criteria
OR
SelectionSpace
GUI as default interface, textual input as ascii file on demand : FunctionalRequirement
GUI: FunctionalRequirement
Ascii-based textual Interface: FunctionalRequirement
InterfaceSelection: Selection
„A GUI is more intuitive than a textual interface, but does not support configuration by means of textual scripts. The choice enables both and respects the higher importance of end users.“ : Rationale
Selection

Figure 4: Exemplary product structure documenting the selection from alternatives

6 Architectural Design

We now introduce the part of the product model which documents architectural designs of a system. As mentioned in Section 3 this product model basically underlies the principle of decision oriented design.

Usually the possible number of alternatives is very large. Therefore the possible optimum alternative often cannot be found due to time and cost restrictions. However the construction of alternatives in a decision-oriented approach is supported by some heuristics like *guidelines***,** *frameworks* or *templates*. These represent principles which in history lead to good decision alternatives and therefore also support the derivation of new alternatives. We add these elements to the basic elements of decision-oriented approaches of Section 3. In the following product model for architectural design we define a product type for each of the elements of decision-orientation. Table 2 shows the mapping between the elements and the product types. The product types themselves will be explained in the succeeding subsections.

Decision Element	Design Product
Alternatives	Architectural Alternatives
Decision criteria	Nonfunctional Requirements
Classification	ArchNFR Evaluation
Rationale	Design Rationale
Solution	Software Architecture
Guidelines, frameworks or templates	Architectural style

Table 2: Decision elements and related model elements for architectural design

In literature (cf. [SG96, IABG97, JGJ97]) the most common understanding of software architecture is, that it represents a hierarchical decomposition of a software system into subsystems. Each of these subsystems defines some unit of the software system and is interconnected via interfaces to other components by predefined communication mechanisms. Despite this common understanding usually these terms are used in very different contexts. A new approach therefore defined a formal model basing on FOCUS (cf. [BS]) to get a precise understanding of these technical terms (cf. [BRS+99]).

Most commonly software architectures aim at the fulfillment of functional requirements with respect to certain nonfunctional requirements by using some general design principles for the decomposition of the software system. In this context Boehm (cf. [SG96]) talks about an intermediate abstraction between user needs and final system structure which helps us to bridge the gap between user requirements and the software system. Others motivate the purpose of software architectures by their support for some pre-selected nonfunctional requirements (cf. [JGJ97]) like support development, maintenance, reuse and evolution of a software system.

In our sense a software architecture consists of both the decomposition of the system into its subsystems as well as a mapping from applied design principles to the resulting subsystems. Design principles are described in our product model by architectural styles, which will also be described more detailed in the succeeding section.

We should accentuate here that in our sense this mapping of an intermediate abstraction to the final system structure is no sufficient motivation for the existence of a software architecture. For a decision oriented design also the decision rationale and the corresponding decision criteria, i.e. why a specific design principle has been chosen, have to be documented as well.

7 Model Elements for Architectural Design

The architectural design part of the product model is shown in the diagram of Figure 5. Note that the products *Architectural Alternative*, *Software Architecture* and *Architectural Style* are software architectures in our sense and therefore can be described with an architecture description. However, they are different in their role in the decision oriented design. For each role different additional aggregations or associations are necessary for documenting the complete content of a product. To demonstrate this difference we defined separate products.

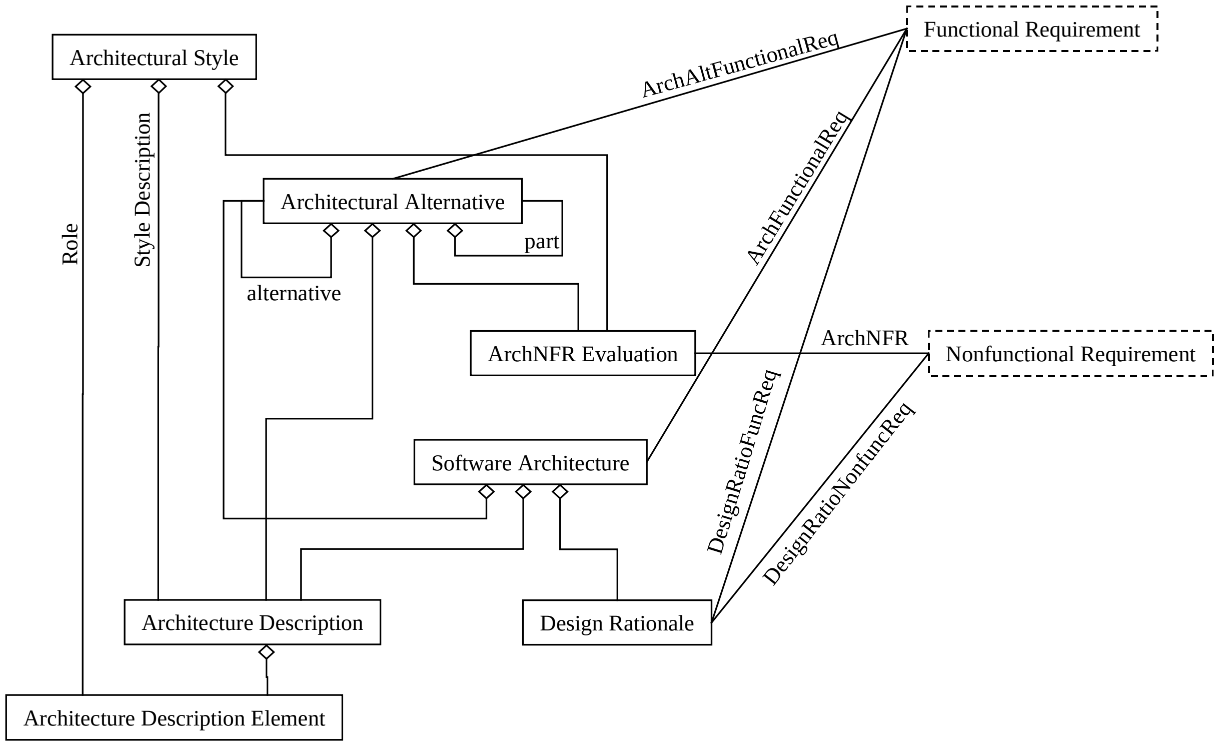

Figure 5: Product structure for architectural design

7.1 Architectural Alternative

The construction of architectural alternatives in the decision oriented design depends mainly on functional requirements and experienced design principles like patterns, known algorithms etc. Appropriate patterns for example are selected and instantiated with specific requirements. Further already existing system parts must be taken into account and usually reduce the possible space of architectural alternatives. We assume that the final software architecture is a composition of architectural alternatives.

Note that we do not assume that only requirements which have been negotiated finally can be used for the derivation of architectural alternatives. In a final solution, however, all functional requirements that have been chosen to be realized in the requirements specification must also be realized in the software architecture. The same applies to nonfunctional requirements. In this case, however, the fulfillment of a requirement is not absolute. Different nonfunctional requirements mutually influence each other. Therefore in order to find a final design solution the different nonfunctional requirements have to compromise.

Product	Architectural Alternative
Description	Each *Architectural Alternative* describes an alternative for a part of a software architecture which shall fulfill a subset of requirements. Each functional requirement which shall be fulfilled by an architectural alternative is expressed by an association between the requirement and the architectural alternative. For example communication between components can be fulfilled by different communication protocols. Each of these protocols represents an architectural alternative. To indicate that the different protocols represent the same subset of requirements, they are aggregated via an *alternative*-aggregation to an *Architectural Alternative* representing all requirements for communication.
Purpose	*Architectural Alternatives* with their part- and alternative- aggregations span a space of possible software architectures for a given set of functional requirements. Each *Architectural Alternative* with aggregated alternatives represents a part of the architecture where there is a possible choice. It supports the selection of an optimal architecture with respect to given functional and nonfunctional requirements.
Attributes	-
Aggregations	Instantiations of part- and alternative-aggregations are mutually exclusive. An alternative cannot have alternative direct sons. part: The actual *Architectural Alternative* is decomposed into several parts. Functional requirements of the actual *Architectural Alternative* are distributed accordingly. alternative: Each aggregated *Architectural Alternative* represents a possible Architecture fulfilling the functional requirements of the *actual Architectural Alternative*. The technical realization of the functional requirements varies between the alternatives. Consequently also the fulfillment of corresponding nonfunctional requirements is differently. Architecture Description: describe the technical realization of the functional requirements. ArchNFR Evaluation: aggregates *ArchNFR Evaluations* with respect to nonfunctional requirements which are relevant for the given *Architectural Alternative*. Relevant are all nonfunctional requirements which are general to the whole software system and all requirements which are related to functional requirements which shall be realized within the given architectural alternative.
Dependencies	-

Association	ArchAltFunctionalReq
Description	Describes the mapping of a functional requirement to an *Architectural Alternative* which shall realize it.
Purpose	The association provides traceability of functional requirements to their realization and backwards.
Attributes	-

7.2 Architecture Evaluations with respect to Nonfunctional Requirements

To build an optimal software system functional as well as nonfunctional requirements have to be fulfilled. Functional requirements can be implemented directly. The fulfillment of pure nonfunctional requirements, i.e. requirements which cannot be operationalized to functional requirements, however, has to be evaluated with respect to a given architecture. The following product serves for documenting this evaluation.

Product type	ArchNFR Evaluation
Description	The associated architecture is evaluated with respect to the classification of the associated *Nonfunctional Requirement*.
Purpose	Provides a methodological and structured basis for documenting the differences between *Architectural Alternatives*.
Attributes	Value: is undefined or a peculiarity of the classification in the associated *Nonfunctional Requirement*. An undefined value means that the given *Architectural Alternative* cannot be judged with respect to the given nonfunctional requirement.
Aggregations	-
Dependencies	-

Association type	ArchNFR
Description	Maps an *ArchNFR Evaluation* to a *Nonfunctional Requirement*.
Purpose	The association shows which *Nonfunctional Requirement* has been evaluated by the given *ArchNFR Evaluation*.
Attributes	-

7.3 Design Rationale

The design rationale is derived from combining nonfunctional requirements and weighting them with respect to priorities of functional and nonfunctional requirements.

Product type	Design Rationale
Description	Describes the reasons for the selection of the *Software Architecture*, why this software architecture has been chosen. The design rationale bases on a prioritization of assigned functional and nonfunctional requirements.
Purpose	The *Design Rationale* is especially important to document design decisions to be able to assess impacts of future evolutions of the software system.
Attributes	-
Aggregations	-
Dependencies	-

Association type	DesignRatioNonfuncReq
Description	Relates a *Design Rationale* to the *Nonfunctional Requirements* which have been taken into account for a specific architectural choice.
Purpose	Assigns all *Nonfunctional Requirements* to a *Design Rationale* which have been taken into account for an architectural decision.
Attributes	-

Association type	DesignRatioFuncReq
Description	Relates a *Design Rationale* to the functional requirements which have been taken into account for a specific architectural choice.
Purpose	Assigns all *Functional Requirements* to a *Design Rationale* which have been taken into account for an architectural decision.
Attributes	-

7.4 Architectural Style

Architectural Styles represent experienced knowledge for good architectural design in history and support the derivation of new architectures.

Product type	Architectural Style
Description	Description of a generic architecture, which is independent of a specific software system and of specific functional requirements. It represents approved principles for architectural design. The parts of an *Architectural Style* get specific roles. Each role fulfills a general task or some predefined general constraints within the design principle. Assigning the roles of the design principle to parts of the final software system can make a mapping between the final software system and an applied design principle. Each *Architectural Style* can be pre-evaluated with respect to some general nonfunctional requirements. *Architectural Styles* comprise different levels of granularity starting at basic principles like modularization (cf. [Par72]) until to complex predefined application specific architectural frameworks. For example the widely known Model-View-Controller-Style (cf. [BMR+96]) consists of three predefined roles for parts of the software: the model, the view and the controller. In a final software system where this style has been applied, parts can be identified which are concrete instances of these roles.
Purpose	Supports the construction of *Architectural Alternatives* and *Software Architectures*. Holds experiences for a good design in an abstract description.
Attributes	Informal Description: describes informally ideas and principles which belong to the given architectural style. Additionally, roles of corresponding *Architectural Description Elements* are described.
Aggregations	Role: points to *Architectural Description Elements* which are instances of roles in the given style. Different style elements may have different roles within the *Architectural Style* which may be described within the informal description of the *Architectural Style*. If an *Architectural Style* is used in a concrete architecture a mapping must exist between all style elements and the *Architectural Description Elements* of the concrete architecture. Style Description: describes the properties of the style elements of an architectural style ArchNFR Evaluation: each *Architectural Style* can be judged with respect to given general nonfunctional requirements. This shall support the selection of architectural styles during the definition of the software architecture.
Dependencies	-

7.5 Software Architecture

Product type	Software Architecture
Description	Concrete architecture of a software system. Represents the unique decision of alternative architectures. The *Software Architecture* is the basis for all further development steps.
Purpose	Documents the complete concrete architecture and the rationale which lead

	to this decision.
Attributes	
Aggregations	Consists of exactly one *Architecture Description* and exactly one *Design Rationale*. Architecture Description: description of the *Software Architecture*. Design Rationale: rationale for the *Software Architecture*. Architectural Alternative: selected *Architectural Alternatives* for the derivation of a software architecture.
Dependencies	-

Association	ArchFunctionalReq
Description	Describes the mapping of a *Functional Requirement* to a *Software Architecture* which shall realize it.
Purpose	The association provides traceability of *Functional Requirements* to their realization and backwards.
Attributes	-

7.6 Architecture Description

Architectural alternatives, architectural styles and the final software architecture have to be described in an appropriate way. For this purpose we defined the products *Architecture Description* and *Architectural Description Element*. Architectural description elements may be refined in an adequate way, like component, interface, connector, protocol, architectural style etc.

Our main purpose, in a first step however, is to describe only the interrelationships between products of requirements definition, project management and design. Therefore it suffices to define only these very general products.

Product type	Architecture Description
Description	The *Architecture Description* describes the constituents of a concrete architecture and their relationships.
Purpose	Represents a typical description of a software architecture.
Attributes	Informal description: describes the architecture informally.
Aggregations	Consists of at least one aggregated *Architectural Description Element*.
Dependencies	-

Product type	Architectural Description Element
Description	Generic element of an *Architecture Description*. Usually related to a great variety of sub-types.
Purpose	Describes a specific part of an architecture description. An *Architectural Description Element* can be connected to project management activities where units of work have to be defined for planning time and resources. They are also the atomic units which may be assigned to roles defined within the description of Architectural Styles.
Attributes	-
Aggregations	Depends on specific sub-type.
Dependencies	-

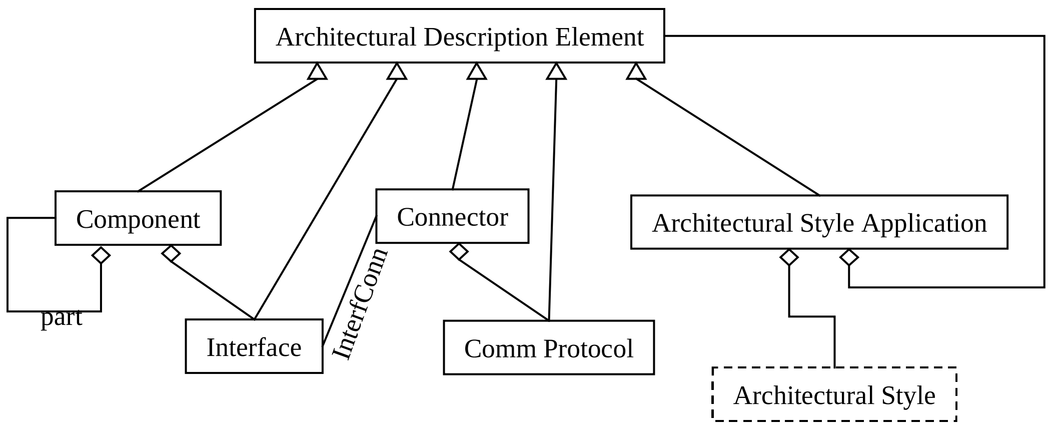

Figure 6: Refinement of Architectural Description Elements

Figure 6 shows an example refinement of *Architectural Description Elements*. We shortly sketch here the basic ideas of these elements. The elements mainly base on the formal model for componentware in [BRS+99]. The previously described *Architectural Style* also is a subtype of *Architectural Description Elements*.

Note that these refinements are only examples for *Architectural Description Elements*. Depending on the specific context subtypes can be added, deleted or exchanged.

Product type	Component
Description	Basic building blocks of a software architecture.
Purpose	Provides the possibility to decompose software systems into separate units.
Attributes	-
Aggregations	part: a *Component* can be decomposed in sub-components. interface: a *Component* may have several interfaces which offer features to their connected components.
Dependencies	-

Product type	Interface
Description	Describes functionality and features which a *Component* offers to other *Components*.
Purpose	To provide the information hiding principle, interfaces are used to separate the internal technical implementation (glass-box) view from the external feature (black-box view). The information hiding principle in general supports the independent development of different components of a software system.
Attributes	-
Aggregations	-
Dependencies	-

Product type	Connector
Description	A *Connector* connects two *Components* via Interfaces. Especially it defines in which way the components communicate.
Purpose	A *Connector* provides every information which is needed to fully describe the communication mechanism which underlies the communication between two interfaces.
Attributes	Communication Parameters: defines parameters which adjust a parameterizable communication protocol.
Aggregations	Communication Protocol: defines, which communication protocol is used. Only one *Communication Protocol* can be aggregated to a connector.
Dependencies	-

Association type	InterfConn
Description	Connects an *Interface* to a specific *Connector*. *Interfaces* may be connected to several *Connectors* at once.
Purpose	*Interfaces* can only be associated directly to *Connectors*.
Attributes	-

Product type	Comm Protocol
Description	A communication protocol describes a specific mechanism for the exchange of messages between communication partners.
Purpose	To describe communication mechanisms independently from concrete connections, communication protocols are described separately.
Attributes	Protocol Description: Describes informally or formally the features of the communication protocol.
Aggregations	-
Dependencies	-

Product type	Architectural Style Application
Description	Maps *Architectural Description Elements* to *Architectural Styles* which have been applied during the derivation of an architecture.
Purpose	Documents the application of *Architectural Styles* within an architecture. In this way it enables traceability of applied principles for a design and documents, how given nonfunctional design decisions have been fulfilled.
Attributes	-
Aggregations	Architectural Style: applied architectural style. An *Architectural Style Application* can only be mapped to one architectural style. Architectural Description Element: Assigns the *Architectural Description Elements* to the predefined roles of the *Architectural Styles*.
Dependencies	-

8 Project Management

This section describes the project management product model. Generally, project management includes the activities project initialization, project planning, project controlling and project ending. This interpretation is increasingly insufficient, because the process-view on project management is not as flexible as needed especially for software development. Therefore, we suggest emphasizing on the product model to determine only a project frame on which the process should be adapted.

Our project management product model covers all resulting products throughout the whole project. We do not cover the prephase in which a customer selects the software developer or the phases after the project such as maintenance. In this context, project management mainly bases on the following tasks:

- **Considering the project frame:** The project frame describes all restrictions and given facts of the project which can either not be influenced by the project responsibles or which have to be negotiated with the customer and therefore are depending on his agreement. In Section 9.1 we will illustrate which products determine the project frame.
- **System and project decomposition:** Complex software systems can be seen as a collection of simpler units, i.e. subsystems, components or modules. Each of these units can be handled easier than the whole system. Therefore it is important to gradually divide and decompose the system into smaller units until the overall project is getting manageable. Generally, there are a lot of methods for decomposition. However, we suggest a stepwise refinement to iteratively provide more detail. During this decomposition the producible elements can be identified to which finally activities can be assigned. The result structure together with the related activities defines the project structure that is reflected within the work breakdown structure. In Section 9.2 you will obtain a detailed overview which products are needed to build up an adequate project structure.
- **Scheduling the project:** The project structure itself is not enough to plan and control the project in order to reach all defined goals; it only simplifies the project. However, to complete the project within the given amount of time and resources it is indispensable to plan and schedule all necessary activities and to estimate the needed effort, resources and costs. Only by defining the activities realistically these conditions can be kept and the project can be completed successfully. Equally important is the strict compliance of the plan and schedules; this is the responsibility of the project manager. In Section 9.3 we describe all products and their relations, which are necessary to plan the project and thus enable a controlling and steering.

In the following we give an overview over all management products which are required to successfully perform and complete the mentioned project tasks. In relation to this, we will continue describing each product of the product structure according to the scheme introduced in Section 2.

9 Model Elements for Project Management

In this section all project management products and their relations illustrated in Figure 7 are described. According to the project management tasks mentioned in the previous section we describe the products and their associations in detail in the following sections.

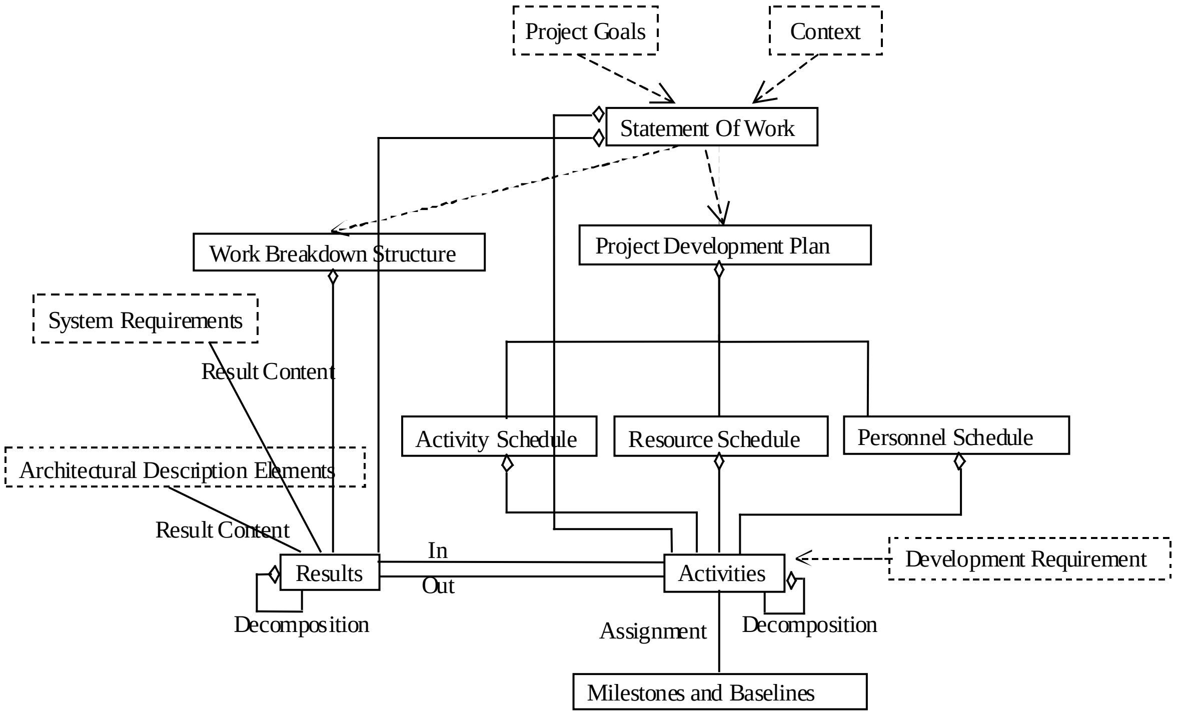

Figure 7: Description of all project management products and relations.

9.1 The Project Frame

Each project consists not only of influencable but also of given or negotiable parts. Influencable parts mean all project parameters which can be directly modified by the project responsibles whereas negotiable parts describe all results and activities which can only be modified if both the project responsibles and the customer agree to them. Both parts strongly depend on the project size and organizational structure. In some cases for example people can be employed to increase the manpower whereas in other cases this is a fix parameter for the project. However, each project should be aware of the static parameters because they strongly influence the scope of action. In this section you will find all products which are connected to the project frame.

Product type	Project Goals
Description	This product shall define unambiguously the project goals. Only in this way, it is achievable to reach a clearly defined end of the project. The goal definition results from the negotiation between the customer, the supplier and the developer. We must take into account, that project goals have to be reachable, without conflicts, easy to examine, unambiguously and adequate to completely fulfill the customer's requirements (cf. [Fib99]). Project goals can refer to contents, costs or schedules. An example of a project goal could be that the software system has to be developed according to a certain process standard.
Purpose	Project goals are the criteria for evaluation of the success of a project.
Attributes	External Responsible: The customer has to determine a responsible role that can decide whether the project goals are adequate.
Aggregations	-
Dependencies	Statement of Work: The project goals are negotiated between the customer, the developer and the project management. They describe the general agreement for the project end. These goals should be adequately reflected and noted in the statement of work.

Product type	Project Context
Description	In this product all information concerning the project context is summarized; that is all externally determined constraints for a project. These parts vary according to each specific project and company. Generally, the project context differs from nonfunctional requirements as it does not mainly refer to issues concerning the system to be developed. Instead, it focuses on organizational aspects to provide basic conditions which facilitate the realization of the project without conflicts. An example could be the hardware infrastructure of a company. Within a large project it is in most cases possible to directly influence or adapt the infrastructure to the project requirements. Especially in large companies this can be done by shifting hardware units from one project to another (cf. [ABD+99]). In opposite to this, the infrastructure is often a fix parameter for small companies where the budget is more restricted and a shifting as described is not possible.
Purpose	The project context determines all unchangeable constraints that have to be considered in order to adequately plan and perform the development project.
Attributes	-
Aggregations	-
Dependencies	Statement of Work: The project context determines the basic conditions such as available resources or the team composition. These restrictions influence the general project scope. Therefore the binding results and activities in the statement of work have to be appropriately defined or adapted.

Product type	Statement of Work
Description	The statement of work (SOW) describes the activities and results required by the customer and agreed to by the developer. The SOW is a formal document which should be incorporated into the contract. The binding version of the SOW is finalized during the contract negotiation. This implies that there can be no binding work items that were informally understood, or agreed to verbally, which do not appear in the SOW. This condition idealistically prevents misunderstandings and disagreements later, after the project begins. An extract of a SOW could include the following results and activities: requirements specification, architectural description, system description, software deliverables, training and testing, etc. Generally, the structure and content of a SOW differs from one development project to another. For example not every project includes the delivery of hardware components, and not all projects require training or installation.
Purpose	The SOW is part of or at least the basis for the contract between the customer and the developer. It describes all obliging main project elements in terms of results and activities that have to be performed or delivered, respectively, to adequately fulfill the project goals.
Attributes	External Responsible: It is necessary for the customer to determine a responsible role who has the power to negotiate and to decide which relevant activities have to be performed during project runtime.
Aggregation	Activities: The SOW aggregates a list of all binding activities. Results: The SOW also aggregates a list of all binding results.
Dependency	Project Development Plan: All activities which have to be carried out must be included and scheduled in the project development plan. WBS: All results mentioned in the SOW must be included in the WBS. Note, this ensures that all results are adequately considered in the project structure.

9.2 System- and Project Decomposition

One quality criterion of each project is the capability to organize and structure work. This is done by the work breakdown structure. The required results are derived from the project context, –goals and the SOW and they are gradually refined to a level of detail which makes it possible to define activities which match and can be assigned to project roles and actors. The needed products and their interconnections are sketched in the following.

Product type	Work Breakdown Structure
Description	The work breakdown structure (WBS) is used to decompose the software project into its basic results and finally activities. Generally, the *system* is structured during the requirement analysis and the design phase from different point of views. Depending on the granularity and the progress of the development new requirements or design elements arise or disappear. Each of these elements has to be represented within the WBS and assigned to at least one activity to produce them. A WBS thus describes the *project* view on the *system.* Possible structuring elements are offered in the DOD standard 2167 for example where the system is mainly divided into segments, computer system configuration items, computer software components and finally computer software units. This order complies with the level of abstraction of the associated elements. Reasonably, these elements should correspond to the architectural description elements resulting from the system design.
Purpose	The work breakdown structures the project into smaller units and thus simplifies it and makes it manageable. Additionally, the WBS describes all results and the associated activities which have to be performed in order to completely realize the system. Above all, it can be used as a controlling instrument as one can examine the overall project *state* according to the state of the incorporated results. For each element in the work breakdown structure, the relationships with architectural description elements and system requirements are described; therefore this product offers transparency about the overall project. The WBS is mainly used as a basis for project controlling as it gives a complete overview of the overall project state by aggregating the state of each partial result to one whole system state. Furthermore it has to be refined and to be updated gradually according to the arising of new requirements and new architectural description elements resulting from the different development phases.
Attributes	-
Aggregations	Results: The WBS aggregates hierarchies of results which are used to structure different project parts such as the requirements specification or the architecture from a management point of view.
Dependencies	-

Product type	Results
Description	Results are developed during the project and are produced through the performed activities. Existing results can then be used as prerequisites for other activities which can only be carried out if they are available, i.e. a subsystem can only be integrated, if all components are present. Generally, the coarse-grained results are defined during the requirements engineering and they are further refined during the design phase. Each result has a state-attribute which characterizes its actual state. In most cases, it is helpful to standardize all results in order to ease operations such as the evaluation of the overall project state.

	Often results and the related activities are assembled to work packages which are then be assigned to a specific role or actor.
Purpose	Results are needed to hierarchically decompose the overall project and to assign the system building blocks to activities.
Attributes	Description: This is a short description of the activity. Type: This attribute enumerates the result type. Possible ones are: reports, architectural elements, source code documents, system requirements, etc. Responsibility: Here, the responsible role has to be assigned. State: This attribute enumerates the state of a result. Possible states for an architectural description element which is to be realized could be: • initial • in progress • in review and • completed. Based on these states, operations can be applied to calculate the state of a set of states or finally the overall project state. Category: Each result should include a unit which allows the use of metrics. In this case we suggest the specification of the following metric categories for effort estimation: • Off-the-shelf: These results have already been developed as part of previous projects. • Full experience: Similar results have already been developed in the past. • Partial experience: These results are in part similar to products which have already been developed in the past. • New development: No experience how to develop this result is present.
Aggregation	Decomposition: Results can be structured according to a logical hierarchical relationship. We must be aware, that this logical structure does not necessarily coincide with the timely order in which the results are to be produced. Here decomposition means, that a hierarchically higher result is achieved, if all its subresults are achieved. Moreover, the attribute values of an aggregating result are dependent on the according values of its subresults. A decomposition hierarchy of results, which we related with system requirements and architecture description elements, respectively, has to be consistent with the relationships between system requirements and the software architecture. For example, a result being related with a given functional requirement is consistently decomposed by results being related with the architecture description elements, that realize this requirement.
Dependency	-

Association type	In
Description	This association describes which activities require the related results. This is important to avoid stagnation which could arise if an activity requires one or more immature results. An example could be an integration activity which needs subsystems to integrate them to the overall system.
Purpose	This association documents consistency conditions that enable to plan the activity reasonably and consistently.
Attributes	Activities: A set of activities to which this association is connected. Results: A set of results to which this association is connected.
Aggregation	-
Dependency	-

Association type	Out
Description	This association describes which results are produced by the related activities. This is important to avoid stagnation which could materialize if the output of one activity which is simultaneously the input of another one has delay.
Purpose	Similar to the In-association, this association also documents consistency conditions that enable to plan the activity reasonably and consistently.
Attributes	Activities: A set of activities to which this association is connected. Results: A set of results to which this association is connected.
Aggregation	-
Dependency	-

Directed Association type	Result Content
Description	This association relates “technical” information structures about a system to be developed namely system requirements and architecture description elements with information structures being relevant for management activities, namely results. Thereby, we bridge the gap between information being structured from a management point of view, and information, which is structured from development points of view.
Purpose	A result content association integrates technical development information with management information explicitly, without having to structure these different kinds of information similarly.
Attributes	-
Aggregation	-
Dependency	-

9.3 Scheduling the Project

A realistic project planing and scheduling is a critical success factor. According to [Dem82] most managers are not able to adequately estimate the effort of software development projects. [Rei93] states that the average cost and time overrun is more than 35 percent. One important reason for these overruns is a poor project planning and -estimation. Activities are planned which are underestimated, inconsistent or even wrong. Therefore it is important to provable define products which offer all information to reduce these failure potentials. Some possible products in order to plan and schedule the project adequately are offered in this section.

Product type	Project Development Plan
Description	The project development plan describes *what* activities have to be performed and *when* they have to be carried out. Contently, the project development plan covers activities such as the timely provision of equipment and tools so that they are available to developers when needed. Also described is the availability of staff to perform the development tasks in accordance with the schedule. Furthermore, contingency plans must be provided in the event that project risks materialize. The project development plan is an important project document. It assures that the development of the project is well charted before the related development activities are enacted. Whenever changes arise during the project, the project development plan has to be adapted to reflect the new situation. This includes above all changes of the project’s schedules which are included in the project plan. One standard which describes the formal structure and the content of a development plan is the IEEE standard 1058.1. There the project generally consists of the traditional activities like project management, configuration management, quality management and software development. Additionally security aspects, change management and customer supplier management are also provided.
Purpose	The project plan is used to give all activities a sensible order. The plan should determine to do the right thing at the right time. Based on the project plan the actual *progress* can be stated on which controlling activities can be performed.
Attributes	-
Aggregations	Activity Schedule: Schedules activities. Resource Schedule: Schedules resources. Personnel Schedule: Schedules manpower.
Dependencies	-

Product type	Activity Schedule
Description	The activity schedule is a schedule of activities and their anticipated time of implementation.

Purpose	The activity schedule lists all necessary activities in a time-ordered way to know what to do anytime.
Attributes	-
Aggregation	Activities: Brings all project activities in a temporary order.
Dependency	-

Product type	Personnel Schedule
Description	In the personnel schedule based on the activities the team composition and the team skills are defined and assigned to the available roles and actors. This strongly depends on the required know-how, motivation and availability. Note, availability means to ensure that each project member can be released according to the defined percentage. Particularly, in software development projects scheduling people is an important competence as people are the most important resource and finally the largest cost-driver (cf. [Boe80]).
Purpose	In this product the team size and structure is defined to always have the personal capacity to carry out the scheduled activities.
Attributes	Communication: The communication relations between all team members should be defined; especially if problems arise. Directives: Especially in large projects, the authority to issue directives should be defined to delimit responsibilities, and areas of responsibilities.
Aggregation	Activities: The personnel schedule aggregates activities and assigns attributes such as • roles, • skills and experiences, • responsibilities and • effort to them.
Dependency	-

Product type	Resource Schedule
Description	This product describes all necessary resources which are needed for the performed activities. These resources not only include tools and hardware but also work space or external partners such as subcontractors or vendors.
Purpose	The resource schedule makes sure that all needed resources exist during the overall project runtime even if critical situations arise.
Attributes	-

Aggregation	Activities: The resource schedule aggregates activities and assigns available resources to them such as • adequate work space, • tools, • equipment as well as • vendors and subcontractors
Dependency	-

Association type	Activities
Description	An activity describes what has to be performed to reach a defined goal that is mentioned in the SOW. Activities correlate with the results either since they require or produce them. Guidance on how this must be done is offered by the development requirements. Furthermore, an inherent characteristic is that activities have a strong time relation as they describe the timely progression of the project. Often, it is useful to standardize the activities to make them comparable and to ease the calculation of effort and time. Therefore, defined amounts must be assigned to the activities, i.e. the activities should be gradually decomposed into units of one man month. Similar to the results, activities can also be decomposed. Thereby they range from high level activities such as software development management or risk analysis to low level activities like coding or module tests. The activities should be decomposed until each activity can be assigned to at least one responsible role. A further refinement is not necessary for management tasks.
Purpose	For an adequate project planning, activities have to be defined in order to reach all project goals within the defined time frame.
Attributes	Description: This is a short description of the activity. Start Date: This refers to the date the activity is scheduled to begin. Completion Date: This refers to the date the activity is scheduled to be completed. Dependencies: This refers to activities on which this activity is dependent. Responsibility: This identifies the role which is responsible for this activity. Additionally, a “category” like described in the results can also be specified.
Aggregation	Decomposition: To create hierarchies of activities, they can be composed or decomposed to alternatives or activities as parts of the whole. Usually, there are two refinement alternatives: The system can be either functionally or object-oriented divided (cf. [Lit93]).
Dependency	-

Product type	Milestones and Baselines

Description	Milestones are always visible and comprehensible project activities. Note that everytime it is possible to recognize whether they have been created or not. Based on this, a founded reporting system can be established and the project progress can be estimated for example by illustrating a trend analysis. If the information has to be more coarse-grained, then the milestones can be put together to milestone plans. These plans give the management sufficient information to control and steer the project.
Purpose	Milestones are important events during the project's runtime. They mark the end of important project steps. They are also used as points of payment and measurement of progress on the project and for determining baselines. Baselines thus describe major milestones. The importance of baselines is obvious as often the first project baseline is the approved system requirements specification document as result of the completion of the requirements activity phase; this is called the functional baseline.
Attributes	Milestone or Baseline ID: This is a meaningful identification i.e. a string or number. Description: This is a short description of the milestone or baseline. Completion Date: This refers to the date the milestone or baseline is scheduled to be completed. Dependencies: This refers to milestones or baselines on which this one dependent. Responsibility: This identifies the person who is responsible for this milestone or baseline. Amount: This characterizes the overall amount which is connected with this developed milestone or baseline. Flag: This attribute describes whether the milestone or baseline is completed or not.
Aggregation	-
Dependency	-

Association type	Assignment
Description	Through this association some activities are assigned as milestones and baselines. Note, that one or even more activities can be assigned to one milestone or baseline.
Purpose	This association characterizes the milestones and baselines of the development project.
Attributes	Activities: A set of related activities. Milestones: The related milestone or baseline.
Aggregation	-
Dependency	-

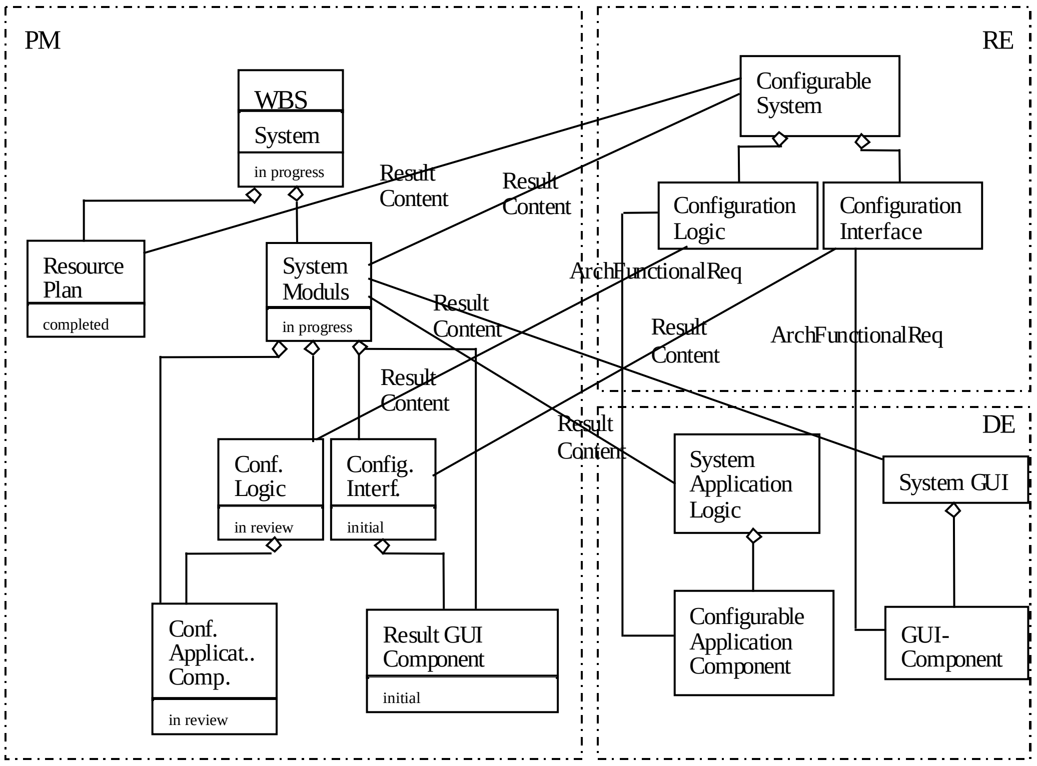

Figure 8: Exemplary work breakdown structure

Finally, Figure 8 exemplifies an instance of a product structure for the development of a configurable computer system (cf. the example given in Section 5.4). In this figure one can see the relationship between requirements, architectural description elements and results of a work break down structure. In our example the major requirement is to produce a system which is configurable. For the realization of the configurable system, a configuration logic and a configuration interface are the identified requirements. For this realization the shown architecture is developed. Thereby, two global design elements are needed, namely the system application logic and the system GUI. Each of these is further subdivided into a configurable application component in case of the system application logic and a GUI component in case of the system GUI. Both are associated with the requirements through an ArchFunctionalReq association.

Additionally, this interconnected development information is linked with the management information in the work break down structure. Results covered in the work break down structure are related with requirements and architectural description elements by means of ResultContent associations. Thereby, we achieve an integrated management view on requirements and architectural elements by bringing them in a hierarchical order and implicitly assigning states and responsibilities to them.

For example the architectural element ConfigurableApplicationComponent is associated with result Conf.Applicat.Comp, which is a subresult of the result Conf.Logic. The latter result is not related with an architectural element, but with the functional requirement ConfigurationLogic which is supposed to be realized by the component ConfigurableApplicationComponent. From a management point of view this result hierarchy is sensible because of the state of Conf.Logic, that is in how far the requirement has already been realized, is dependent on the state of the result *Conf.Applicat.Comp*, that is in how far the related component has already been specified and implemented.

Based on this information, we can manage system development adequately. Note that the product structure does not enforce a timely order of building the architecture, defining requirements, and building the work breakdown structure. In any way we only must respect the defined causal dependencies.

Now, we summarize some project management aspects arising with the use of the described product model:

First, the work breakdown structure is based on the high level results coming from the requirements analysis and the design. These elements are gradually put into the WBS and in that way hierarchically ordered. This promotes the integration of technical and organizational issues as well as a clear structuring of the project which eases the manageability. Note, that the integrated results do not have to be reflected in every detail, instead it is sufficient to include results which can be assigned to at least one responsible role or actor. In this *ResultContent* association the relationship between the management results and the corresponding requirements and architectural description elements respectively is illustrated.

Second, the described product model allows dynamical alteration of the system and the project as well. Based on the consistency conditions mainly described in the “results” and “activities” products and the “in”, “out” and “result content” association a flexible adding and deleting of results is possible. This is especially useful for projects with changing requirements – which is widely spread in software development projects.

Third, the separation in results and activities and their representation in the WBS and the activity schedule, respectively eases the planning and controlling of the project. While the former focuses on the project state, the latter focuses on the project’s progress. This complies to two different views which both must be regarded during a project’s lifecycle. If, for example, the management ascertains a deviation between the planned and the actual progress, then it is not only interesting to find out which activities are responsible for this delay, but also to recognize which system parts are concerned. Based on this knowledge the management is able to perform some steering measures in order to realign the project. These measures could for example include a consistent and customer accepted reduction of functionality.

Fourth, as a result of the strongly coupled result hierarchy related with requirements, architectural elements and project management results, a common information basis is established. This is exemplified in Figure 8 as there is a strong network of ArchFunctionalReq and *ResultContent* associations between all development parts. Although, some results are not represented as requirement or design element, nevertheless, there are associations which show their relationships.

Fifth and last but not least, the WBS lists all results and their interrelations in order to describe the causal dependencies between the different results. This is for example useful when the management wants to estimate the remaining effort and therefore needs information about the result stream. For a detailed analysis it is even possible to follow the links to technical details, if the management wants to obtain a precise knowledge of specific and corresponding requirements or architectural aspects.

10 Comprehensive Dependencies

In the previous sections we already discussed relationships between elements of the product models for the three covered tasks. Here we want to emphasis on three dependencies which couple the three models tightly. These dependencies relate three key work products, namely requirements specification, software architecture and work break down structure. Essentially the dependencies define that all requirements contained in the requirements specification must be reflected in the software architecture, and that structuring a project's work from the management point of view must refer, at least partially, only to requirements and architectural description elements contained in the requirements specification and the software architecture, respectively.

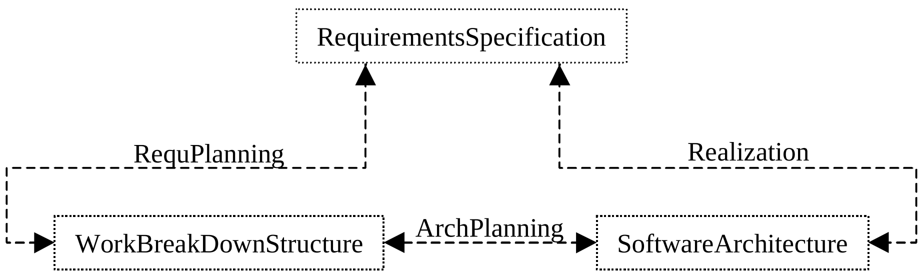

Figure 9: Dependencies between requirements specification, software architecture and work break down structure

Below, the dependencies illustrated in Figure 9 are described in a schema similar to the one for associations. In order to admit top-down as well as bottom-up development, we defined the dependencies to be bi-directional.

Dependency type	RequPlanning
Description	Structuring a project's work in a work break down structure is partially based on requirements. For example, when a system's architecture is not yet known, we can structure and plan future work on selected requirements. A requirements specification and a work break down structure are consistent, if all the system requirements to which a result of the work break down is associated (by a ResultContent association) are contained in the requirements specification. Note, that not all system requirements of a requirements specification need to be associated with a result. For example, for planning purposes it can be sufficient to only relate a given system requirement explicitly to a result, but not its sub-requirements (aggregated by an AND refinement).
Purpose	This dependency defines a consistency condition between a work break down structure and a requirements specification. It ensures that planning is based only on those requirements, for which we decided that they have to be realized by the envisioned system.

Dependency type	Realization
Description	A requirements specification and a software architecture are consistent, when all requirements contained in the requirements specification, have been "realized" (functional) or "respected" (nonfunctional) in the architecture. A functional requirement is "realized" by a software architecture, if the requirement is associated with the architecture via an *ArchFunctionalRequ* association. A nonfunctional requirement is "respected" by a software architecture, if the requirement is associated with the architecture's *Design Rationale* via a *DesignRatioNonfuncReq* association.
Purpose	This consistency condition ensures, that finally all negotiated and selected requirements have been respected in the software architecture.

Dependency type	ArchPlanning
Description	Structuring a project's work in a work break down structure is partially based on the software architecture. A software architecture and a work break down structure are consistent, if all the architectural description elements to which a result of the work break down structure is associated (by a ResultContent association) are contained in the software architecture.
Purpose	This dependency defines a consistency condition between a work break down structure and a software architecture. It ensures that planning is based only on those architectural description elements, for which we decided that they are part of the software architecture.

11 Conclusion and Outlook

In this paper we presented a model of work products for the requirements, architectural and project management parts of software development. This provides a basis for integrating process models of different application domains as well as different development tasks. In order to reach a clear structure of the model we introduced firstly templates for work product types, associations, aggregations and dependencies between them. Then we introduced the principle of decision oriented software development process and related general products.

By applying the templates for work products of requirements engineering, design and project management first we showed the applicability of the templates. Further the work product model defines the basic skeleton for a development process model which integrates these different tasks. Further we related requirements engineering and design work products to the general products of the decision oriented development process.

In the product structure for requirements engineering we defined work products allowing us not only to formulate the final result of requirements engineering tasks, but also to support taking and documentation of decisions. Furthermore, we sketched the relation between domain modeling and specification of requirements. We classified requirements in a way being well suited for design purposes.

The product structure for design firstly documents a decision oriented design process by work products for supporting the derivation and definition of architectural alternatives, evaluating alternatives, documenting the rationale and the final result. Secondly it consists of structures for describing software architectures themselves.

For project management, we defined a model based on which we can plan and control project enactment and progress. We structure a project's work by means of activities and results. We understand certain results as management views on system requirements and architectural description elements, respectively. Thereby we achieve a close coupling of management and development information.

The work products of requirements engineering and design are integrated in two ways. Functional requirements within the requirements specification relate directly to parts within the final architecture. In contrast to this nonfunctional requirements relate to architectural decisions, i.e. depending on the content and on the priority of the given nonfunctional requirements different architectural alternatives have to be selected.

We also integrated project management with the technical tasks requirements engineering and architecture. The main integration is done by using requirements as a first basis for planning activities of the work breakdown structure. When a project proceeds and the system structure becomes more detailed the architectural elements are related to the results leading to a more detailed project structure.

The whole model for work products can directly be used for development projects. Its main advantage is that it not only defines clearly the information which has to be gathered but also makes it possible to check consistency and completeness of work products.

Currently we plan to evaluate the product model with examples from different application domains within our research consortium FORSOFT. Directly related to this is also a refinement of the work product model with respect to these application domains. A future aim is the integration of the work product model with development activities to provide a full and very flexible software development process model.

12 References

[ABD+99] D. Ansorge, K. Bergner, B. Deifel, N. Hawlitzky, C. Maier, B. Paech, A. Rausch, M. Sihling, V. Thurner, S. Vogel: Managing Componentware Development – Software Reuse and the V-Modell Process, CAISE, 1999.

[BDRS97] M. Broy, E. Denert, K. Renzel, M. Schmidt (Eds.): Software Architectures and Design Patterns in Business Applications. Technischer Bericht TUM-I9746, Technische Universität München, 1997.

[Ben94] E. M. Bennatan, Software Project Management – A Practitioner's approach, McGraw-Hill Book Company, 1994.

[BLR+95] A. Bröckers, C.M. Lott, H.D. Rombach, M. Verlage: MVP-L language report version 2. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1995.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, "Pattern – oriented Software Architecture – A System of Patterns", Wiley, 1996.

[Boe80] Garry Böhm: Software Project Management. Addison Wesley, 1980.

[BRS+99] K. Bergner, A. Rausch, M. Sihling, A. Vilbig, M. Broy: A Formal Model for Componentware. in Foundations of Component-based Systems, eds. M. Sitaraman, G. Leavens, Cambridge University Press, to appear 1999.

[BS] M. Broy, K. Stølen, "FOCUS on System Development", to appear.

[Dei98b] B. Deifel: Theoretische und praktische Ansätze im Requirements Engineering für Standardsoftware und Anlagenbau. Technischer Bericht TUM-I9832, Technische Universität München, 1998.

[Dem82] Tom De Marco: Software-Projektmanagement. Wolfram's Fachverlag, 1982.

[Fib99] Projektmanagement Fibel, http://www.managementsoftware.de/, 1999.

[JBR98] I. Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process. Addison Wesley Longman, 1998.

[JCJ+92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: Object-Oriented Software Engineering. A Use Case Driven Approach. Addison Wesley, 1992.

[Jeus92] M. A. Jeusfeld: Change Control in Deductive Object Bases. INFIX Pub, Bad Honnef, Germany, 1992.

[JGJ97] Jacobson I., Griss M., Jonsson P., "Software Reuse", Addison Wesley Longman, 1997.

[Kru98] Philippe Kruchten: The Rational Unified Process. Addison Wesley, 1998.

[KS98] G. Kotonya, I. Sommerville: Requirements Engineering. John Wiley & Sons, 1998.

[Lit93] H.-D. Litke: Projektmanagement – Methoden Techniken und Verhaltensweisen. Hanser Verlag, 1993.

[LK95] P. Loucopoulos, V. Karakostas: System Requirements Engineering. McGraw-Hill, 1995.

[LT96] R.C. Linger, C.J. Trammell, Cleanroom Software Engineering Reference Model. Technical Report, CMU / SEI-96-TR-022, Pittburgh, Pa.: Carnegie Mellon University, Software Engineering Institute, November, 1996.

[LW98] A. van Lamsweerde, L. Willement: Inferring Declarative Requirements Specifications from Operational Scenarios. In IEEE Transactions on Software Engineering, Vol. 24, No. 12, December 1998.

[MBJ+90] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: Telos: Representing Knowledge about Information Systems. Transactions on Information Systems 8, 4 (1990), pp. 325-362.

[MCY99] J. Mylopoulos, L. Chung, E. Yu: From Object-Oriented to Goal-Oriented Requirements Analysis. In Comm. of the ACM, Vol. 42, No. 1, January 1999.

[Pae98] B. Paech: The Four Levels of Use Case Description. in 4th Int. Workshop on Requirements Engineering: Foundations for Software Quality, Pisa, June1998, E. Dubois, A. Opdahl, K. Pohl (eds.), 1998.

[Par72] D.L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", Comm. ACM, Vol. 15, Num. 12, Dec. 1972.

[PCW+93] M. Paul, M. Chrissis, C. Weber, S. Shrum: A Preview of the Software CMM Version 2, http://www.sei.cmu.edu/pub/cmm/v2/cmm-v2-bridge.pdf, 25.08.1999.

[Pohl96] K. Pohl: Process-Centered Requirements Engineering. Research Studies Press Ltd, JohnWiley & Sons Inc, New York,1996.

[Rei93] Donald J. Reifer: Software Management. IEEE Computer Society Press, 1993.

[RG94] C. Rolland, G. Grosz: A General Framework for Describing the Requirements Engineering Process. In Proceedings of the Int, Conf. on Systems, Man and Cybernetics, San Antonio, TX, October 1994. IEEE Computer Society Press, 1994.

[RJB99] J. Rumbaugh, I. Jacobson, G. Booch: The Unified Modeling Language Reference Manual. Buch, The Addison-Wesley Object Technology Series, Addison-Wesley, 1999.

[RRP99] J. Ralyté, C. Rolland, V. Plihon: Method Enhancement with Scenario Based Techniques. Proceedings of the Conference on Advanced Information Systems Engineering CAiSE'99, LNCS 1626, pp.103-118, 1999, Springer-Verlag, 1999.

[SG96] Shaw M., Garlan D., "Software Architecture – Perspectives on an emerging discipline", Prentice Hall, 1996.

[VB96] Vossen G., Becker J.: Geschäftsprozeßmodellierung und Workflow-Management. Int. Thomson Publishing GmbH, 1996.

[IABG97] IABG: V-Modell 97. http://www.iabg.de, 1999.