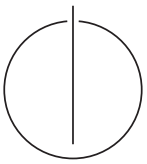
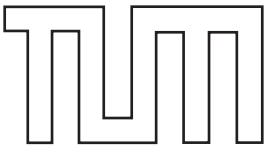


TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Software & Systems Engineering
Prof. Dr. Dr. h.c. Manfred Broy



# SPES 2020 Deliverable D1-1

## Model-based Development

Motivation and Mission Statement of Work Package *ZP-AP 1*


Author: Alarico Campetelli, Martin Feilkas,
Martin Fritzsche, Alexander Harhurin,
Judith Hartmann, Markus Hermannsdörfer,
Florian Hölzl, Stefano Merenda,
Daniel Ratiu, Bernhard Schätz,
Wolfgang Schwitzer

Version: 1.0
Date: October 23, 2009
Status: Completed

Technische Universität München - Fakultät für Informatik - Boltzmannstr. 3 - 85748 Garching

# Document History

| Author | Content | Date | Version |
|---|---|---|---|
| Daniel Ratiu | Initial Version | | 0.1 |
| Judith Hartmann | Initial Version reworked | | 0.1 |
| Alexander Harhurin, Martin Fritzsche | Review | | 0.1 |
| Daniel Ratiu | Review comments integrated | | 0.2 |
| Alexander Harhurin, Martin Fritzsche, Flo Hölzl, Alarico Campetelli, Markus Herrmannsdörfer | Review | 17.03.2009 | 0.2 |
| Florian Hölzl | Reworked Fig. 1 explanation | 17.03.2009 | 0.2.1 |
| Judith Hartmann | Review comments integrated | 17.03.2009 | 0.3 |
| Daniel Ratiu | Released | 21.07.2009 | 1.0 |

## Contents

# 1 The Challenge of Developing Embedded Systems

Software is the most important innovation enabler for technical systems today. By software we realize new functions, we find new ways of implementing known functions with reduced costs and/or higher quality. Above all, what is in particular important, is that with the help of software we can easily combine functions and correlate them into multi-functional systems. Due to its increase in pervasiveness and importance, the software will take the dominant role in the development of many technical systems.

However, the development of complex software for embedded systems poses a variety of challenges classified below along three dimensions: firstly, there are challenges faced due to the increased size and complexity of the software and that are common to all software producers; secondly, there are challenges due to the economic impact and cost model of technical systems; and above all, there are superimposed challenges due to the specifics of embedded systems.

**Increasing size and complexity** Increasing size due to a multitude of different functions and increasing complexity due to their extensive interaction are challenges that must be faced during the development of complex software.

- **Increasing amount of software.** The amount and importance of software in technical systems is growing exponentially. For example, today we find in premium cars over ten millions of lines of code and in the next generation we expect to find ten times more; the Airbus A380 contains over one billion lines of code. In comparison, Microsoft Windows XP has about 40 million lines of code.
- **Complex and subtle dependencies.** Complex systems are made of sub-systems that interact with each other many times only in highly subtle ways. The software is used to drive each of these sub-systems and thereby the software deployed in a sub-system needs to interact with one deployed in another sub-system. In the case of such multi-functional-systems, functionalities are not isolated of each other, but heavily interact with each other.
- **High need for reuse.** In order to address challenges like shorter time-to-market, cost pressure, and high quality, there is a strong demand for the reuse of software artifacts. However, systematically developing high quality reusable software components is non-trivial and has influences throughout the whole development process.
- **Product variants.** In order to address the customer needs and the technology changes, technical domains (e. g., the automotive domain) are characterized by a big number of different product variants. A higher degree of variability leads to an enormous increase in complexity of both development artifacts and processes.

**Business constraints.** The increasing importance of software has as effect that it is a decisive factor for business success. Time to delivery, costs, and protecting the business competence through patents are typical business factors that affect the software.

- **Time and cost pressure.** Due to the high dynamics on the market, the period of time in which a system needs to be delivered is decreasing. At the same time, in order to remain competitive, the costs need to be reduced. The reduction of time-to-market and costs demands in turn high reuse, global optimizations, and increased automation.
- **Technology and user expectations changes.** Both technologies and user expectations change at light-speed. Practically every few years there is a major improvement to be taken and every decade happens a major technology change. Many systems are regarded as outdated and thereby lose their business value. It is primarily the software systems that need to keep up with these changes.
- **Coordination between the supplier and integrator.** Most of the complex systems are made of parts that are developed by third parties (suppliers). The relation between suppliers and integrators implies also issues related to protecting intellectual property.

**Embedded systems particularities.** A third class of challenges are specific to embedded systems and are super-imposed on the traditional difficulties of building large and complex software. The new problems arise due to physical constraints (e. g., time-to-response), the need to interact with the mechanical parts, or to the fact that embedded software is mostly used as control software and thereby needs a high reliability.

- **Heterogeneity of systems.** Software covers, influences, integrates, and coordinates a wide variety of parts of a system from mechanical to electrical and logical subsystems. The resulting software, taken as a whole, is highly heterogeneous since it covers a multitude of different gadgets from entertaining systems to navigation and control systems. Many of these systems interact (indirectly) with each other.
- **High distribution.** Embedded software is essentially distributed as it runs on different ECUs that need to communicate with each other in order to implement the desired functionality.
- **High dependability.** Many embedded systems need to meet high dependability requirements such as safety, high availability, or hard real-time constraints.
- **Existing hardware solutions.** The existing hardware solutions are important factors in the development of embedded systems and thereby have a strong impact on the software.
- **Lifecycle mismatch between functionality and technical realization.** In various domains, the development of embedded systems is characterized by a lifecycle mismatch between the realized functionality and its technical realization. Even if large parts of the required functionality stays stable over product generations, rapidly changing technologies and hardware solutions prevent expedient reuse of the implemented software.

We need to remark that many of these challenges influence each other and are sometimes contradictory. For example, the high dependability demands a “worst case engineering” approach while achieving low costs imposes global optimizations and “average case engineering” strategy; technology changes demands brand new products, while time and costs pressure demands a high reuse.

*It is our belief that these challenges can be efficiently addressed by raising the abstraction level at which the development is done today and by using adequate models along the entire development process.*

## 2 Situation in practice

Model-based development is adopted more or less consequently in practical development of embedded systems today. The pervasive use of models allows the engineers to abstract from implementation details, to raise the level of abstraction at which systems are developed, use verification techniques, and to increase the automation. As a consequence, model-based development increases the productivity and quality of software development for embedded systems.

However, model-based development approaches often fall short due to the lack of powerful enough modeling theories, integration of theories, methods and tools. Even if artifacts are modeled explicitly, they are based on separate and unrelated modeling theories (if foundations are given at all), which makes the transition from one artifact to another in the development process unclear and error-prone. Below we enumerate several of the *pressing issues* with the use of model-based development in the practice:

**Lack of semantic foundation:** Many critical issues arise due to a missing, conflicting or inappropriate semantic foundation of the modeling languages used today.

*a) Weakly-defined semantics:* The semantics of modeling languages is many times only specified as prose (if at all) and thus insufficiently defined. This leads to ambiguities in the interpretation of the created models and to inconsistencies between different implementatios of the language in different tools. For example, a consequent usage of UML is difficult since a unified semantic interpretation of its models is missing.

*b) Multitude of dialects of modeling languages:* In praxis, there are modeling languages for which different semantic foundations exists. This results in different dialects of the same modeling technique. The existence of several dialects prevents an uniform treatment of the built models. For example, state machines exist in different dialects in StateMate, UML and Rhapsody. The models written in one dialect are ill-formed in another dialect or require a different semantical interpretation. These facts make the translation of models written in one formalism to another formalism difficult or even practically impossible. Furthermore, engineers that are experienced in one dialect might misinterpret models written in another dialect.

*c) Fragmentation, isolation, and heterogeneity of semantic foundations:* Since we aim at an integrated model-based development, a unified modeling theory is required which is comprehensive enough to support the whole development process starting from initial requirements down to a running system. Instead of this, today, complementing modeling techniques are used to support the expression of different aspects of the system. Even if these languages are semantically founded, it is often the case that their semantic foundations differ. For example, UML 2.0 defines 13 types of diagrams for different development stages (e. g., use-case diagrams, activity diagram, component diagrams, deployment diagrams). Even if there are formalization approaches that rigorously describe individual diagrams, once a system is described in

different diagrams, it is not clear how they can be combined and integrated. This leads to the impossibility to deeply integrate the models since the integration of different models written in different languages requires a uniform semantic basis, and understanding of the relation between different languages.

**No appropriate architectural model:** Today's model-based software development involves the usage of different models at different stages in the process and at different abstraction levels. However, an integrated architectural model (at best enriched with domain specific concepts) is missing which leads to the (logical) isolation of the developed models.

*a) Missing integration of developed models:* Unfortunately, the current approaches do not make clear which kinds of models should be used in which process steps or how the transition between models should be done. The choice of a particular modeling technique to be used is done in an ad-hoc manner and mostly based on the experience of the engineers or on the modeling capabilities of the tools at hand. As a consequence, the developed models are logically isolated. This subsequently leads to a lack of automation, to difficulties to trace the origins of a certain modeling decision along the process, or to perform global analyses that transceed the boundaries of a single model.

The logical isolation of the developed models leads to difficulties with the management of intent and consistency between different models.

Due to the isolation of models, the intent and rationale behind component designs is lost in the process. In order to document the intent more explicitly, trace links between different model elements in different tools are required. A typical example for missing trace links is requirements tracing information that is lost during the transition from a requirements engineering to a design modeling tool. Ideally, a rich and seamless modeling infrastructure should enable the deduction of the intent of artefacts from the comprehensive model.

Consistency problems are twofold: vertical consistency and horizontal consistency. In the vertical case, we need to make sure that the models at two different phases of the development process are consistent to each other (e. g., requirements and design). The horizontal consistency is between the models created in the same phase (e. g., two views showing a perspective of the design). Ideally, a comprehensive architectural model should enable the automatic checking of consistency.

*b) Missing modeling abstractions:* In order to reduce the complexity and the distance between the requirements and the running systems as well as to support reuse, we need to perform incremental steps and to work at different layers of abstraction from requirements, to design, and to deployment. Such a system of abstraction layers is a basis for a systematic design process for for embedded software systems. However, there are currently no architectural models that support the description of a system along different abstraction layers.

*c) Inappropriateness of modeling languages:* Engineers need several years to develop a specific product family and ideally, in order to be efficient, the modeling languages that they use should directly support modeling these products. Today the same modeling languages, methods and tools are used for developing completely different products such as airplanes and cars. The modeling languages are thus highly general and most of the times domain independent and ontologically neutral. This leads to a big conceptual gap between the professional languages

of engineers and the modeling languages. This subsequently leads to weakly defined modeling languages that are too general and "not aware" of the specifics of a domain. Ideally, a modeling language should provide support for creating and manipulating problem-level abstractions as first-class modeling elements.

Consequently, the architectural model should be instantiated with domain specific concepts that allow the engineers to work in direct analogy with their domain knowledge. The modeling abstractions should be appropriate both with respect to the process phase when they are used and to the domain that is modeled.

**Insufficiencies of the engineering environments:** Mature tool support is a central prerequisite for scaling up model-based development. Many problems arise due to tooling insufficiencies. Today, engineers have to use already existent off-the-shelf tools for developing their models. Instead of using appropriate tools, due to various constraints (e. g., licensing prices, unavailability of tools) the engineers need to adapt themselves to the tools at hand. These tools are only weakly integrated with each other (if at all) and this causes redundancies and loss of information and of automation in transition between artefacts.

*a) Lack of suitable tools:* Only very few model-based technologies are backed-up by tools that are robust and powerful enough for industrial use. For example, despite its shortcomings, the success of UML is (arguably) based also on the fact that it is well-supported by commercial tools. Due to the high costs of tool development, only a few companies can afford to build their own solutions. Therefore, there is a general tendency to work with commercial-off-the-shelf (COTS) tools in industrial practice. On the one hand, COTS tools are too powerful and provide functionality that their users do not need in their daily work in a specific project. On the other hand, these tools are most of the times not aware of the specifics of the system being developed and thus do not support their users effectively.

*b) Isolation of tools:* Today many different kinds of tools are used to develop a large system. Most of the times these tools are standalone solutions: they are designed to be used as simple standalone programs that are executed by one developer on one machine. The integration of tools is done today in a peer-to-peer manner using tool couplers. This approach does not scale, since the number of couplers increases exponentially in the number of tools. Moreover, the coupler-based integration of tools is done in an ad-hoc manner in order to solve specific pragmatic problems. Ideally, the tools should be part of an integrated models engineering environment (e. g., much like Eclipse) and the boundaries between tools should be inexistent allowing a seamless transition between models and process phases without loss of information.

## 3 Our Mission

The challenges of developing technical systems could be efficiently approached through the use of adequate models during the development. The real benefits of models would take effect if they are used throughout the whole development process as a development artefact. In seamless model-based development, modeling is not just an implementation method, but it

is a paradigm that provides support throughout the entire development and maintenance life cycle.

The goal of work package *ZP-AP 1* is to develop *the foundations of a comprehensive and seamless approach to the model-based development of heterogeneous embedded systems that comprises the process and tool integration.* Below we explain our goal:

1. *Foundations of model-based development.* We focus on model based development, that means the pervasive use of models along the development phases. This allows the engineers to abstract from implementation details and to raise the level of abstraction at which systems are developed. Through foundations we understand issues that exceed the technical pragmatic horizon of today's development and aim to tackle the paradigm shift that will happen in five to ten years. However, we stress that it is not only our goal to present foundational theories, but instead to develop above a language layer that is adequate for the industry, and approachable by domain experts without a deep mathematical background.

2. *Comprehensive and seamless.* Rather than covering islands of the process, our work should comprehensively encompass all important aspects from all development process phases. In addition to covering all development phases, we focus on the transition between different process steps. We focus on unification of the foundations that should allow to describe different views and to cross the boundaries between different views by using the same framework.

3. *Process and tool integration.* The steps for developing the models should be integrated in a coherent process definition rather than applied in an isolated manner in different phases. Furthermore, the methods should be backed up by tool support that transforms the tools from being islands of automation, into enablers of the pervasive automation that tackle different views of the product.

*Seamless model-based development promises to lift software development onto higher levels of abstraction by providing integrated chains of models covering all phases from requirements elicitation and management to system design, implementation, and verification.*

Modeling starts early in the development process with requirements engineering where informal requirements are turned into models step by step. At the end of the requirements engineering phase, having reduced the flexibility of informal expressiveness while having gained precision by formalization, we have obtained a functional model capturing these requirements. In turn, the system architecture and in sequence the software architecture is described by models that capture different aspects of the system, namely the software application in terms of a logical architecture and its execution environment in terms of a technical architecture (Figure 1 - left). Provided these models are chosen carefully and founded on an adequate modeling theory, the architecture models can be verified to guarantee that the functional requirements are fulfilled by the integrated and accepted system. It is sufficient to verify the (logical) architecture model against the functional requirements and the technical implementation of single components against the component specification in order to ensure the correctness of the overall implementation. An integration test as well as an acceptance test are – at least from a theoretical point of view – dispensable (Figure 1 - right). Nevertheless, they are useful and essential to ensure non-functional requirements as e. g., the usability of the system.

We believe that consequent front-loading of testing and verification activities in combination with a modeling theory supporting composability and property preservation along the line of development results in ideally no effort during the integration and acceptance test phases. Therefore seamless and comprehensive model-based development is a key to a more systematic, disciplined and predictable development process with the potential of a higher grade of automation and a higher development efficiency.

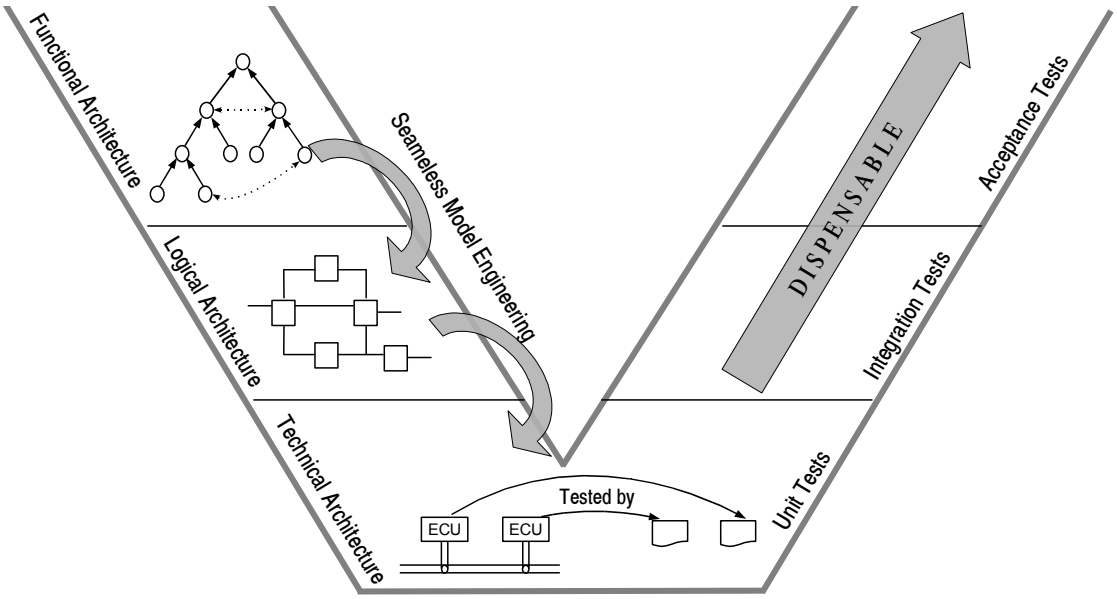



**Figure 1: Model-based Development**

**Our approach.** We tackle our goals by developing in the following directions:

1. a *comprehensive modeling theory* that serves as semantic basis for the models (AP1.1),
2. an *integrated architectural model* that holistically describes the product (product model) as well as the methodical steps to develop it and integrate it into a process (AP1.2),
3. a set of automatic and comprehensive *verification techniques and strategies* to enable the assurance of high dependability standards (AP1.3), and
4. an *integrated model engineering environment* that conforms to the modeling theory and allows the authoring of the product model (AP1.4).

To be able to work out such an approach, a number of ingredients are required as illustrated in Figure 2. These ingredients can be divided into three levels: the *semantic domain* forms the basis for describing the *holistic architectural model* whose construction is operationalized by a *models engineering environment*. In the following, we detail on these levels and their corresponding ingredients.

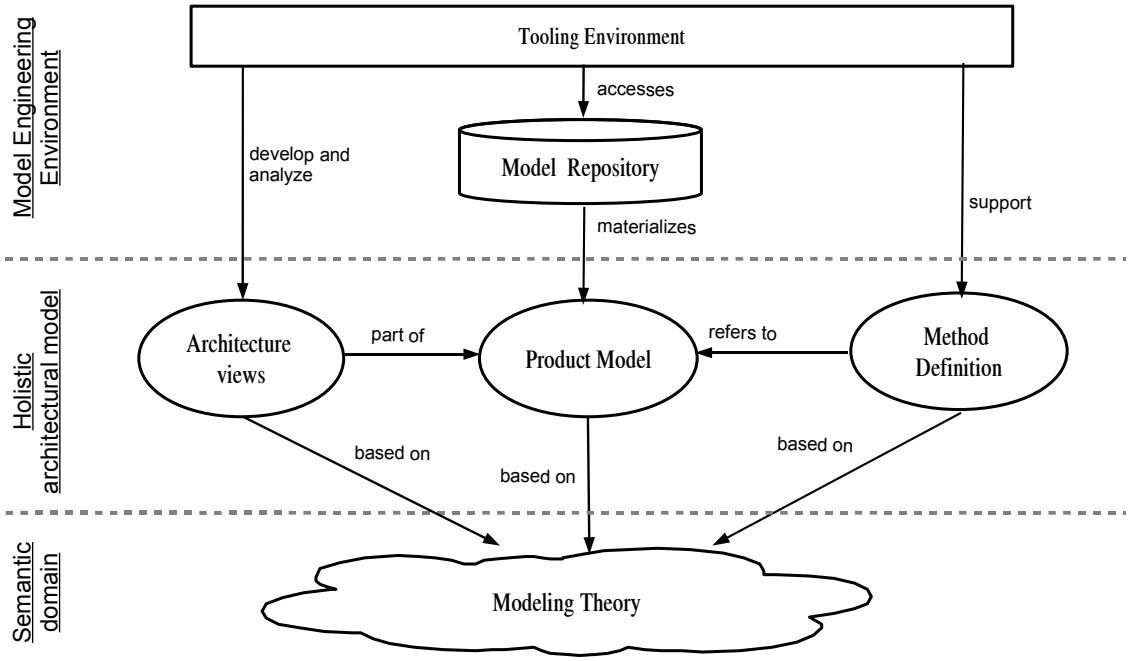



**Figure 2: Main ingredients of seamless model-based development**

### 3.1 Semantic Domain

Seamless model-based development requires a comprehensive modeling theory as basis to ensure a thorough formalization of all artifacts produced during the development of a system. An appropriate modeling theory provides firstly the appropriate modeling concepts such as the concept of function hierarchies, with

1. a concept for structuring the functionality by function hierarchies,
2. concepts to establish dependency relationships between these functions, and
3. techniques to model the functions with their behavior in isolation including time behavior and to connect them – according to their dependencies – into a comprehensive functional model for the system,

and secondly the concept of architectures to capture

1. the decomposition of the system into components that cooperate and interact to provide the system functionalities,
2. the interfaces of the components including not only the syntactic interfaces but also the behavior interfaces, and
3. a notion of modular composition which allows us to define the interface behavior of a composed system from the interface behaviors of its components.

The modeling theory must be strong and expressive enough to model all relevant aspects of hardware and software architectures of a system such as structuring software deployment, description of tasks and threads, as well as modeling behavior aspects of hardware. These

aspects and properties of architecture should be represented in a very direct and explicit way. Our approach to such a modeling theory is given by the FOCUS[1] theory and its various extensions.

### 3.2 Integrated Architectural Model

A comprehensive architecture model of an embedded system and its functionality is the basis for a product model that comprises all the content needed to specify a distributed embedded system. Furthermore, the architectural model needs to be domain-appropriate and thereby to allow the engineers to specify the system in direct analogy to their domain knowledge. In Figure 3 we illustrate our mission to move from today's situation where islands of general models are used in model-based development towards comprehensive and domain-appropriate models.

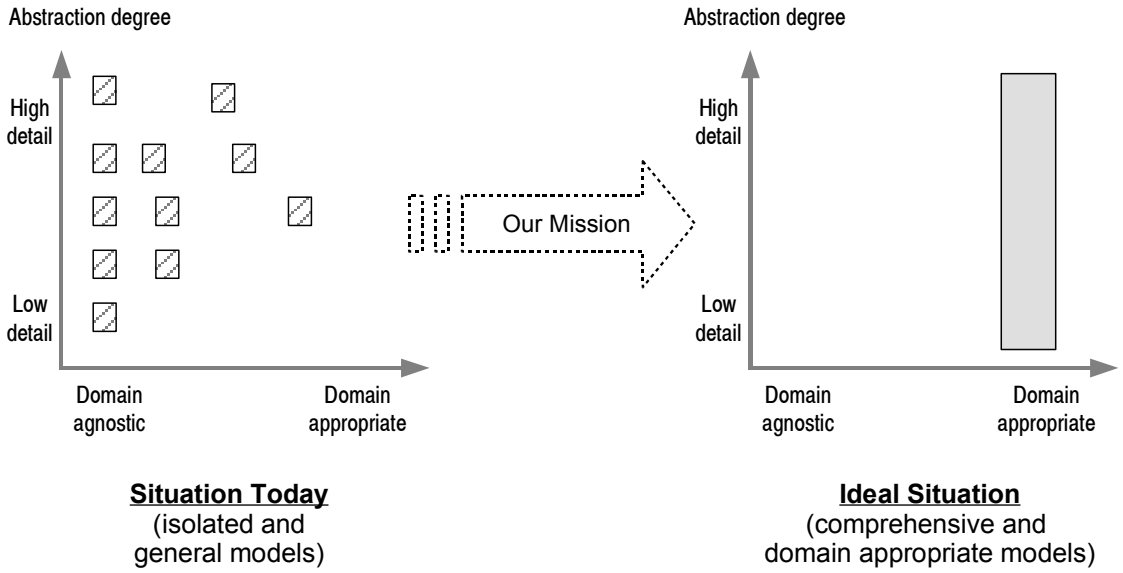



**Figure 3: Comprehensive and appropriate modeling**

In order to develop comprehensive models that cover all process phases (from requirements to implementation) we need different *abstraction layers*. An architectural model describes all model views that define a system at different abstraction levels and the relations among them. It enables a systematic and domain-appropriate development process and represents the starting point for tool support. All views on a system are part of the product model.

### 3.3 Integrated Model Engineering Environment

A central characteristic of model-based development is a high degree of automation by extensive tool support. The level of automation that can be achieved strongly depends on the used models and the associated theory. In Figure 4 we illustrate intuitively our mission to move from the

[1] Manfred Broy, Ketil Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001

current situation where the existent tools dictate the models and theories to be used towards the ideal situation where the tools are implemented such that they can fulfill the needs of the engineers. The support for automation has to address the capturing and elaboration of models, the analysis of models with respect to their consistency and important properties as well as techniques for generating further development artifacts from models. Tooling should be based exclusively on the product model. Then all tools that carry out the steps of building models, analyzing models and generating new artifacts from existing ones basically only manipulate and enhance the product model. The whole development should be regarded as an incremental and iterative process with the goal to work out the contents of a comprehensive product model. In order to turn the vision of high automation into reality, we need an integrated engineering environment that offers support for creating and managing models within well-defined process steps.

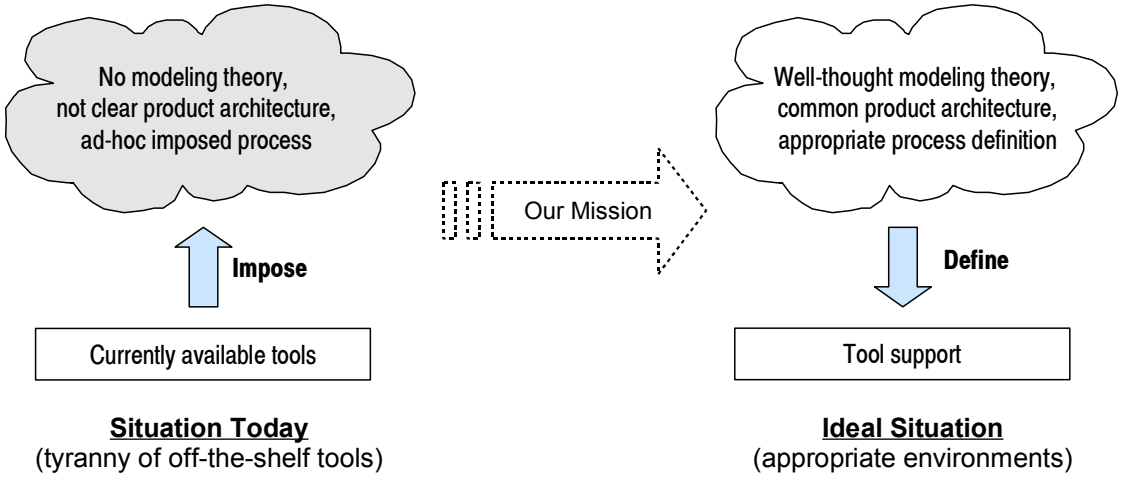



**Figure 4: Appropriate models engineering environment**

## 4 AP 1 Workpackages at a Glance

In Figure 5 we visualize the mapping of the workpackages from AP1.

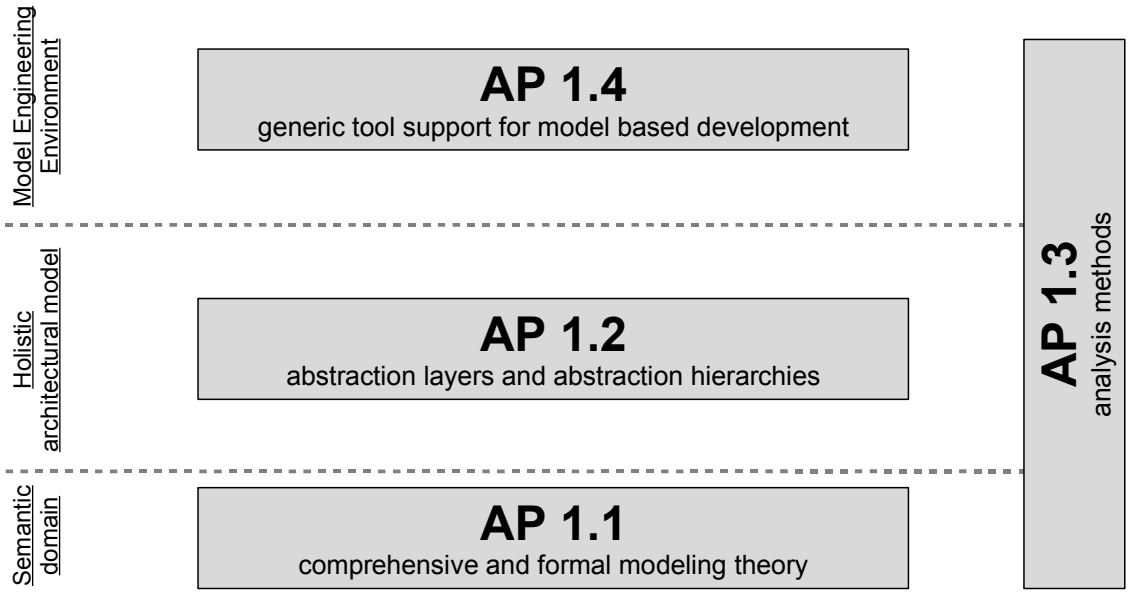



**Figure 5: AP1 workpackages at a glance**