

MDI framework

<!-- -->

Table of contents

1	Requirements of the framework.....	2
1.1	Service-oriented architecture (SOA).....	2
1.2	Adaptive business architectures.....	3
1.3	MDA and adaptive architectures.....	3
2	Specification of the framework.....	4
2.1	Conceptual integration.....	5
2.2	Technical integration.....	7
2.3	Applicative integration.....	10
3	Realisation of the framework.....	12

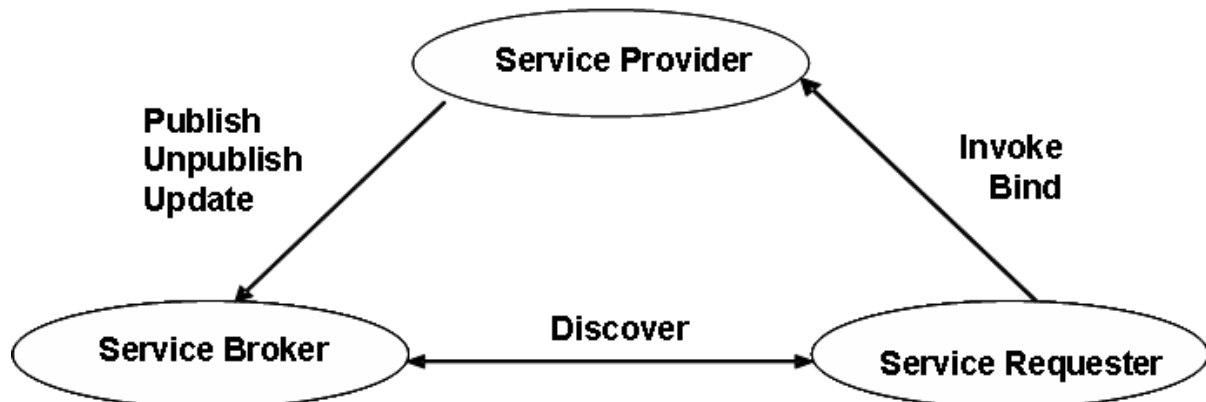
1. Requirements of the framework

We believe that there is a need for an interoperability framework that provides guidance on how MDD should be applied to address interoperability.

The interoperability framework integrates principles of model-driven, service-oriented and adaptive software architectures:

- Model-driven architectures focus on design-time aspects of system engineering. Model-driven development methodologies describe how to develop and utilise (visual) models as an active aid in the analysis, specification, design and implementation phases of an ICT system.
- Service-oriented architecture (SOA) specifies systems composed of services offered by various service providers, which provides the basis for supporting new business models, such as “virtual organisations”.
- Adaptive software architectures focus on run-time aspects of system engineering. Agent and P2P technologies enrich an ICT system with dynamic and adaptive qualities.

1.1. Service-oriented architecture (SOA)



A service-oriented architecture (SOA) is defined by W3C as “A set of components which can be invoked, and whose interface descriptions can be published, discovered and invoked over a network.” (W3C). In a service-oriented model we typically find three roles:

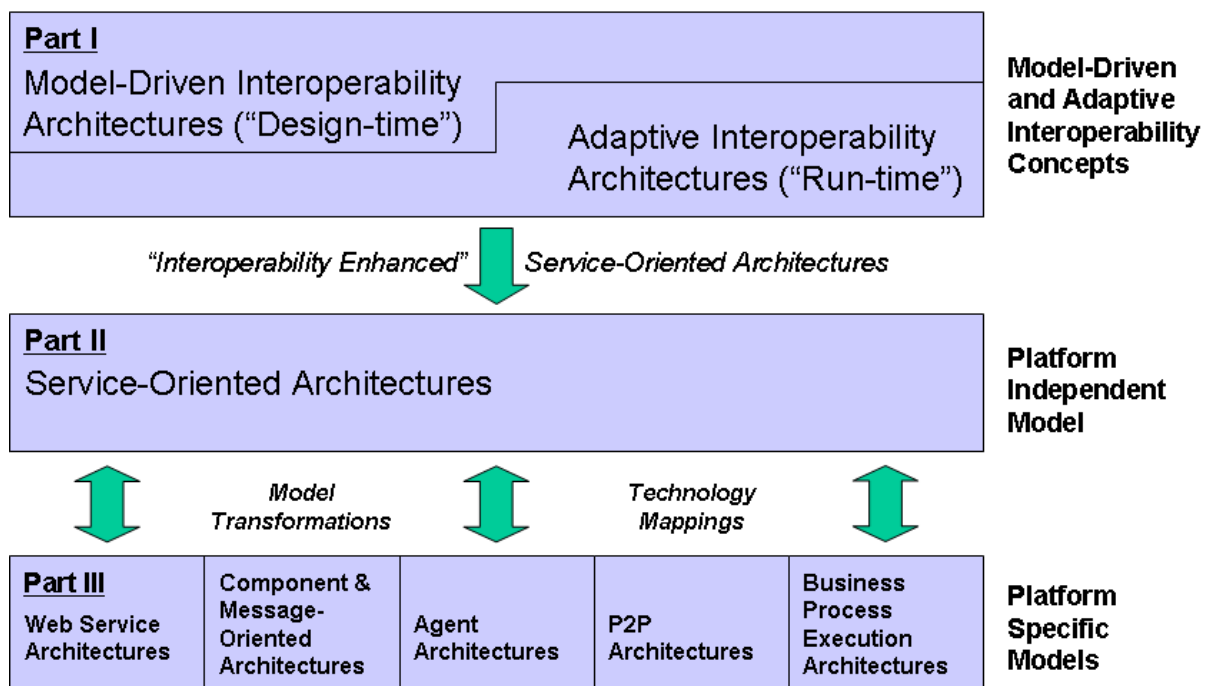
- **Service provider:** A service provider is the party that provides software applications for specific needs as services. Service providers publish, unpublish and update their services so that they are available on the Internet.
- **Service requester:** A requester is the party that has a need that can be fulfilled by a service available on the Internet. A requester could be a human user accessing the service through a desktop or a wireless browser; it could be an application program; or it could be another service. A requester finds the required services via a service broker and binds to services via the service provider.

- Service broker:** A service broker provides a searchable repository of service descriptions where service providers publish their services and service requesters find services and obtain binding information for these services. Examples of service brokers are UDDI (Universal Description, Discovery, and Integration) and XMethods. In many cases the role of the service broker is not explicitly needed. Services can be discovered by marketing channels or by referrals.

1.2. Adaptive business architectures

Globalisation and mass customisation are business trends that have created a situation where business processes that used to be managed by a single enterprise are now distributed and need to be planned and enacted across multiple partners. This leads to highly distributed architectures of business systems with amplified complexity and increased need for collaboration among the participating organizations. At the same time, organizations are in increasing need of solutions to allow them to cope flexibly with constantly changing environments.

1.3. MDA and adaptive architectures



The interoperability framework integrates principles of model-driven, service-oriented and adaptive architectures:

- Model-driven architectures focus on design-time aspects of system engineering. Model-driven development methodologies describe how to develop and utilise (visual)

models as an active aid in the analysis, specification, design and implementation phases of an ICT system.

- Service-Oriented Architecture (SOA) specifies systems composed of services offered by various service providers, which provides the basis for supporting new business models, such as “virtual organisations”.
- Adaptive interoperability architectures focus on run-time aspects of system engineering. Agent and P2P technologies enrich an ICT system with dynamic and adaptive qualities.

2. Specification of the framework

<p><u>Conceptual Integration</u></p> <ul style="list-style-type: none"> • Concepts • Metamodels • Languages • Model relationships 	<p><u>Applicative Integration</u></p> <ul style="list-style-type: none"> • Methodologies • Standards • Domain models
<p><u>Technical Integration</u></p> <ul style="list-style-type: none"> • Development environments • Execution environments 	

The ATHENA model-driven interoperability framework builds on the vision: “Enterprises are able to flexibly develop and execute interoperable applications based on model-driven development approaches to service-oriented and adaptive software solutions”. The framework integrates principles of model-driven development, service-oriented architectures and adaptive architectures. The framework is structured in three main integration areas:

- **Conceptual integration** which focuses on concepts, metamodels, languages and model relationships. It provides us with a foundation for systemising various aspects of software model interoperability.
- **Technical integration** which focuses on the software development and execution environments. It provides us with development tools for developing software models and

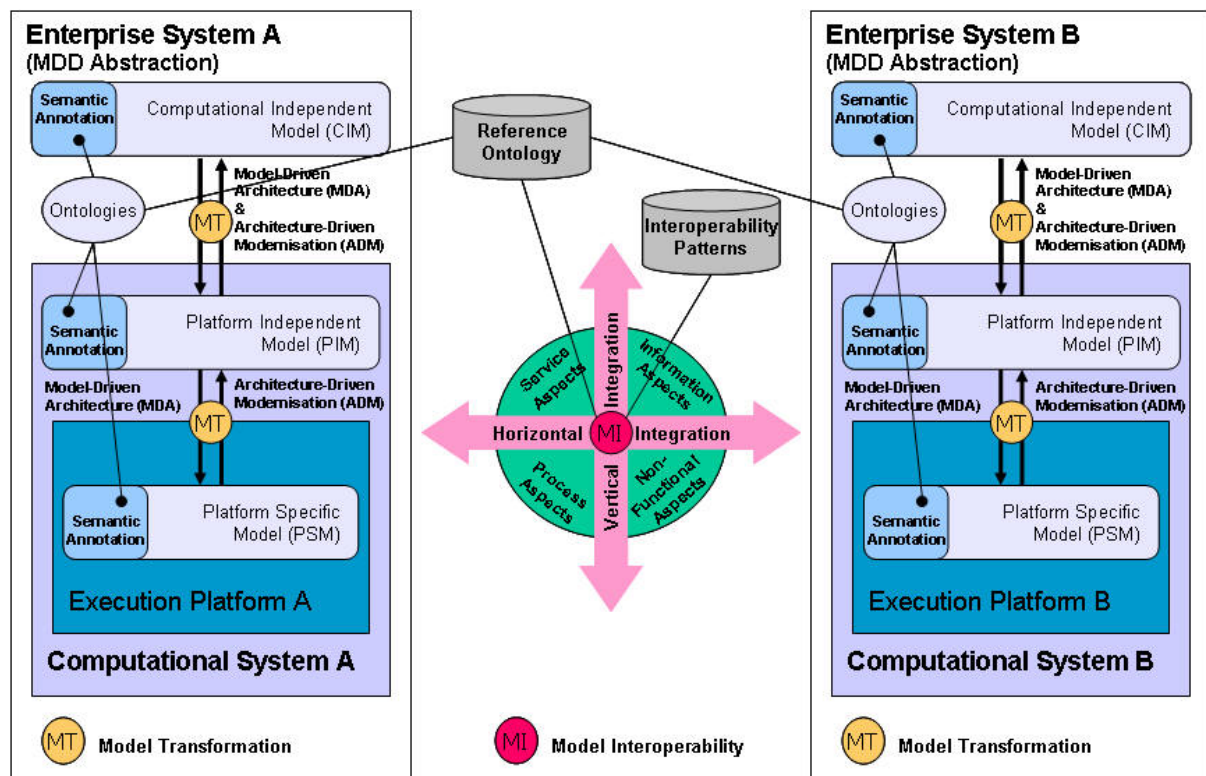
execution platforms for executing software models.

- **Applicative integration** which focuses on methodologies, standards and domain models. It provides us with guidelines, principles and patterns that can be used to solve software interoperability issues.

For each of these three areas a reference model to describe and support the application of model-driven development of software systems is specified.

2.1. Conceptual integration

The reference model for conceptual integration has been developed from a MDD point of view focusing on the enterprise applications and software system.



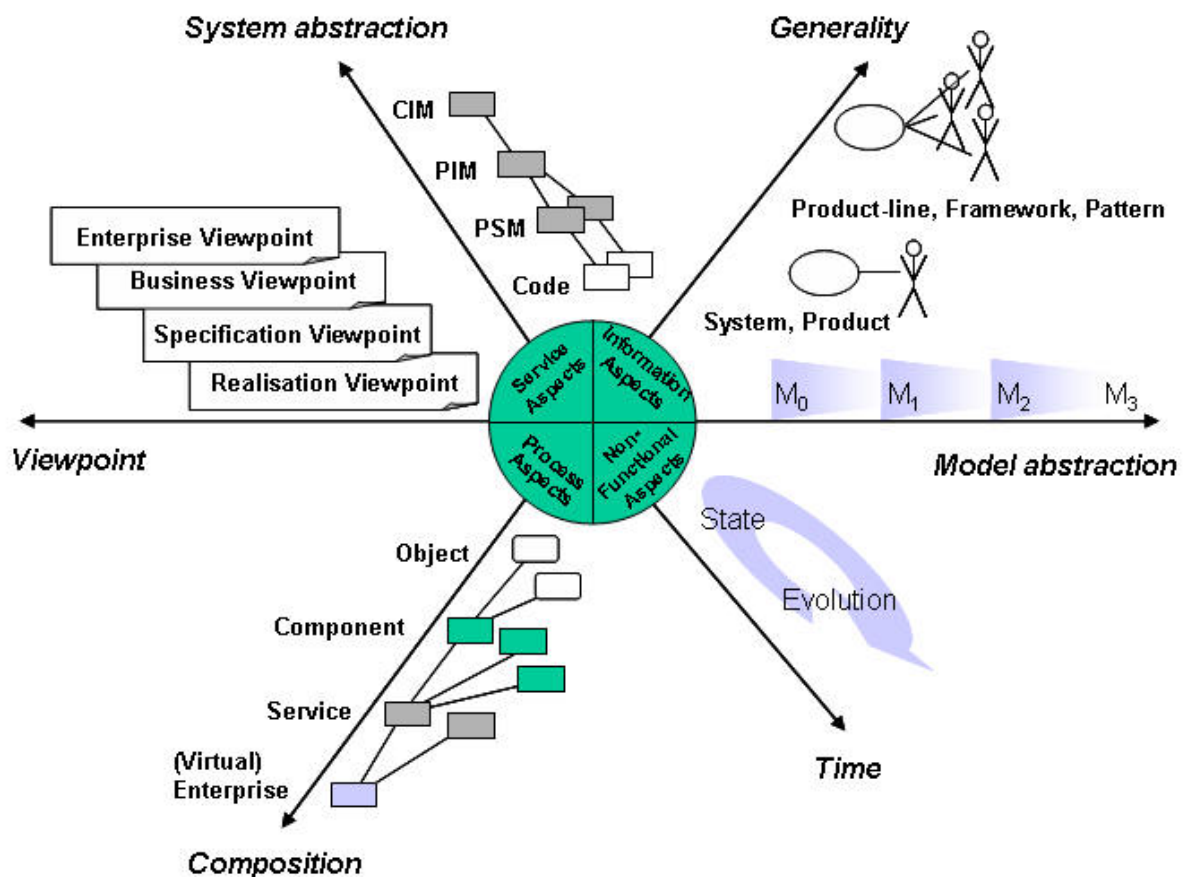
A computation independent model (CIM) corresponds to a view defined by a computation independent viewpoint. It describes the business context and business requirements for the software system(s). A platform independent model (PIM) corresponds to a view defined by a platform independent viewpoint. It describes software specifications independent of execution platforms. A platform specific model (PSM) corresponds to a view defined by a platform specific viewpoint. It describes the realisation of software systems. The figure shows this relationship with respect to an enterprise system. It also shows how MDA and ADM could be perceived as a “top-down” and a “bottom-up” approach to software

development and integration. The models at the various levels may be semantically annotated using reference ontologies which help to achieve mutual understanding on all levels. We also see the usage of interoperability patterns for horizontal and vertical integration.

We have identified four categories of system aspects where specific software inter-operability issues can be addressed by conceptual integration. These four aspects can be addressed at all three CIM, PIM and PSM levels.

1. **Service aspects:** Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity.
2. **Information aspects:** Information aspects are related to the messages or structures exchanged, processed and stored by software systems or software components.
3. **Process aspects:** Processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc.
4. **Non-functional aspects:** Extra-functional qualities that can be applied to services, information and processes.

All of the elements discussed above are integrated into the figure where we look at horizontal and vertical integration between multiple enterprise systems, here exemplified with two enterprise systems A and B.



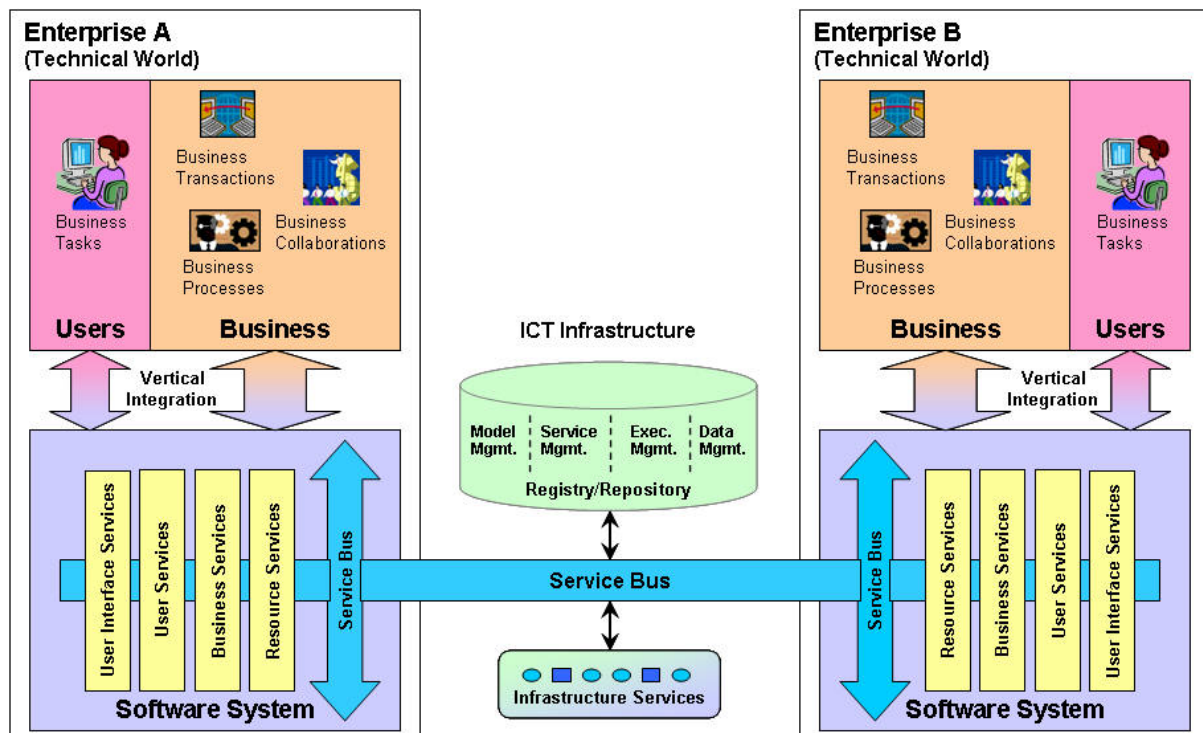
We will use this reference model to address model interoperability, where meta-models and ontologies will be used to define model transformations and model mappings between the different views of an enterprise system.

- **System abstraction:** This dimension of system design reflects the abstraction in terms of implementation independency and is addressed by MDD.
- **Generality:** The applicability of a system design has impact on the adaptability and reusability of the system components.
- **Viewpoint:** System models represent a complex and strongly interrelated network of model entities. To address different issues and for complexity reduction different viewpoint on the model are used. This viewpoint may also be regarded for interop-erability.
- **Composition:** Systems are iteratively composed in a hierarchy from individual objects to the system in the enterprise context. On each of this aggregation layers the entities have to be interoperable.
- **Time:** The system itself is modified in status, configuration and design.
- **Model abstraction:** Meta-models help to describe and analyse the used models.

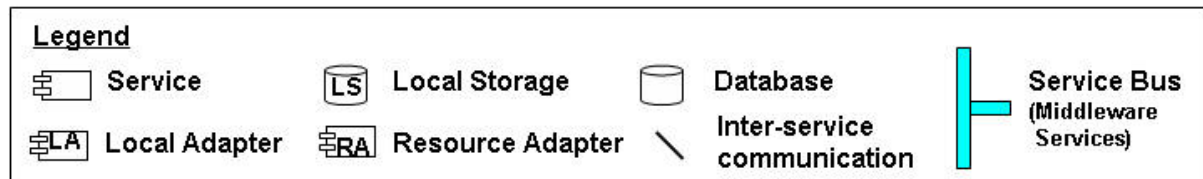
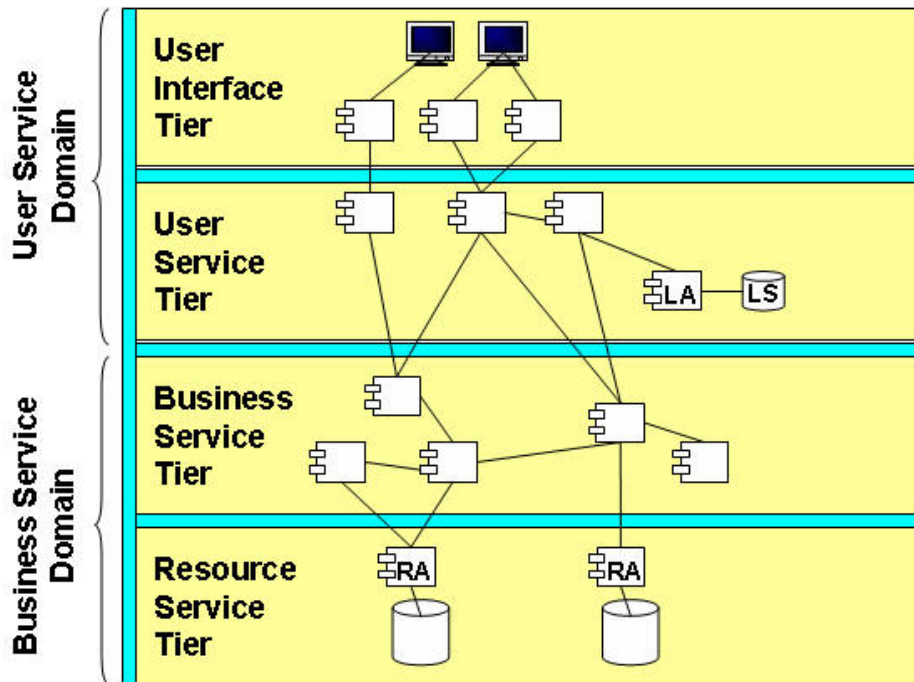
These dimensions can be used to analyse software systems or help to structure the system modelling process and to catalyse design decisions. Each of these dimensions may support interoperability achievements or could represent a challenge of interop-erability.

2.2. Technical integration

The reference model for technical integration has been developed from a service-oriented point of view where a software system provides a set of services required by the businesses and users of the enterprise.



The architecture of the enterprise applications and software systems can be described according to a 4-tier reference architecture where each tier provides different software services required by the enterprise. The software system itself is coupled to an ICT infrastructure illustrated by a service bus that provides the necessary communication infrastructure. Infrastructure services such as composition, mediation, matchmaking and transformation that enables interoperability between software systems should be provided. We recognize the need for a model repository for managing models of various kinds, a service registry for managing naming, directory and location of services, an execution repository for managing information and state needed in the execution of software services and processes, and a data repository for managing results and traces of the executions. The figure shows how a service bus comes into play when integrating two (or more) enterprises systems. The service bus will make use of infrastructure services, and registry and repository.



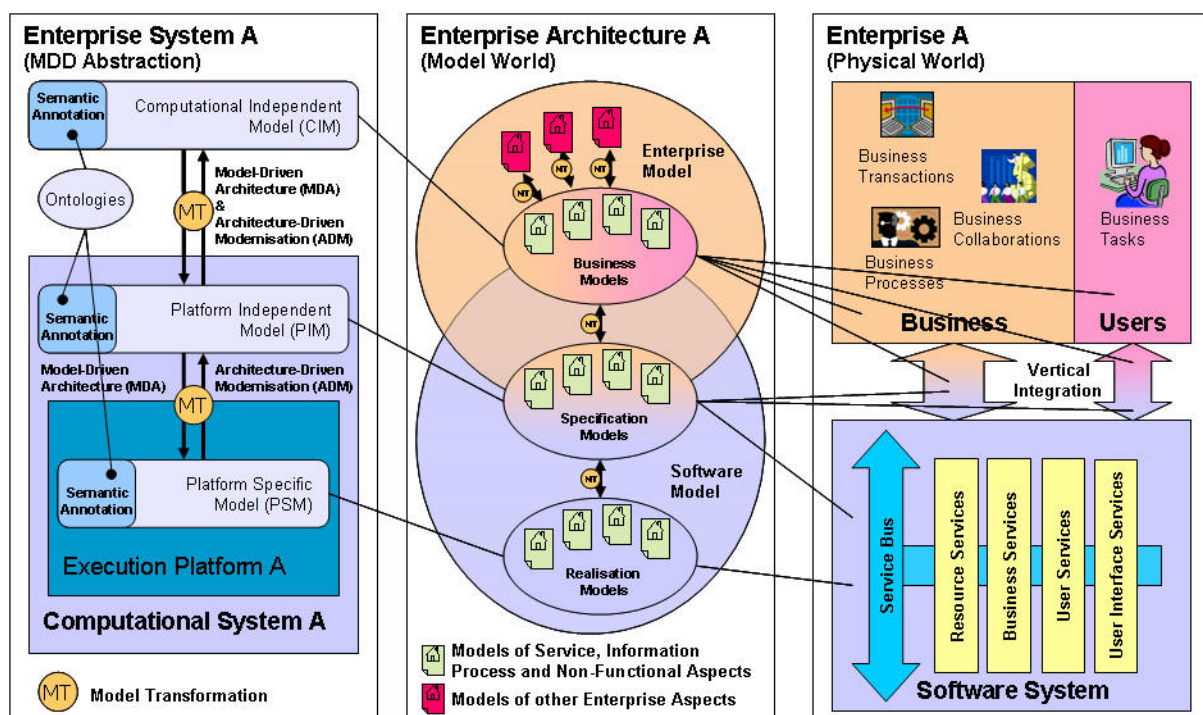
We have defined a reference architecture that separates the architecture of a software system into four logical tiers. The reference architecture consists of a local user-space called the user service domain, and a shared transactional business-space called the business service domain. The four tiers are as follows:

1. **User interface tier** provides presentation and user dialog logic.
2. **User service tier** provides the user's model, which may include user session logic and user-side representations of processes and information. It is an abstraction for a set of business services.
3. **Business service tier** provides components that represent business functionality and pervasive functionality (vertical vs. horizontal services). This tier provides enterprise-level services, and is responsible for protecting the integrity of enterprise resources. Components in this tier can be process-oriented, entity-oriented or work-flow-oriented.
4. **Resource services tier** provides global persistence services, typically in the form of databases. Resource adapters (e.g. JDBC or ODBC drivers) provide access, search and update services to databases and its data stored in a database management system (DBMS) like Oracle or Sybase.

In addition to these four tiers we need a service communication bus so that services deployed at the various tiers can interoperate both within a tier and across tiers.

2.3. Applicative integration

The reference model for applicative integration has been developed based on work related to enterprise architecture frameworks and software architecture frameworks. Enterprise and software models can be related in a holistic view, regardless of modelling language formalisms, by the use of metamodels. This is important in order to understand the dependencies between the different models and views to achieve interoperability.



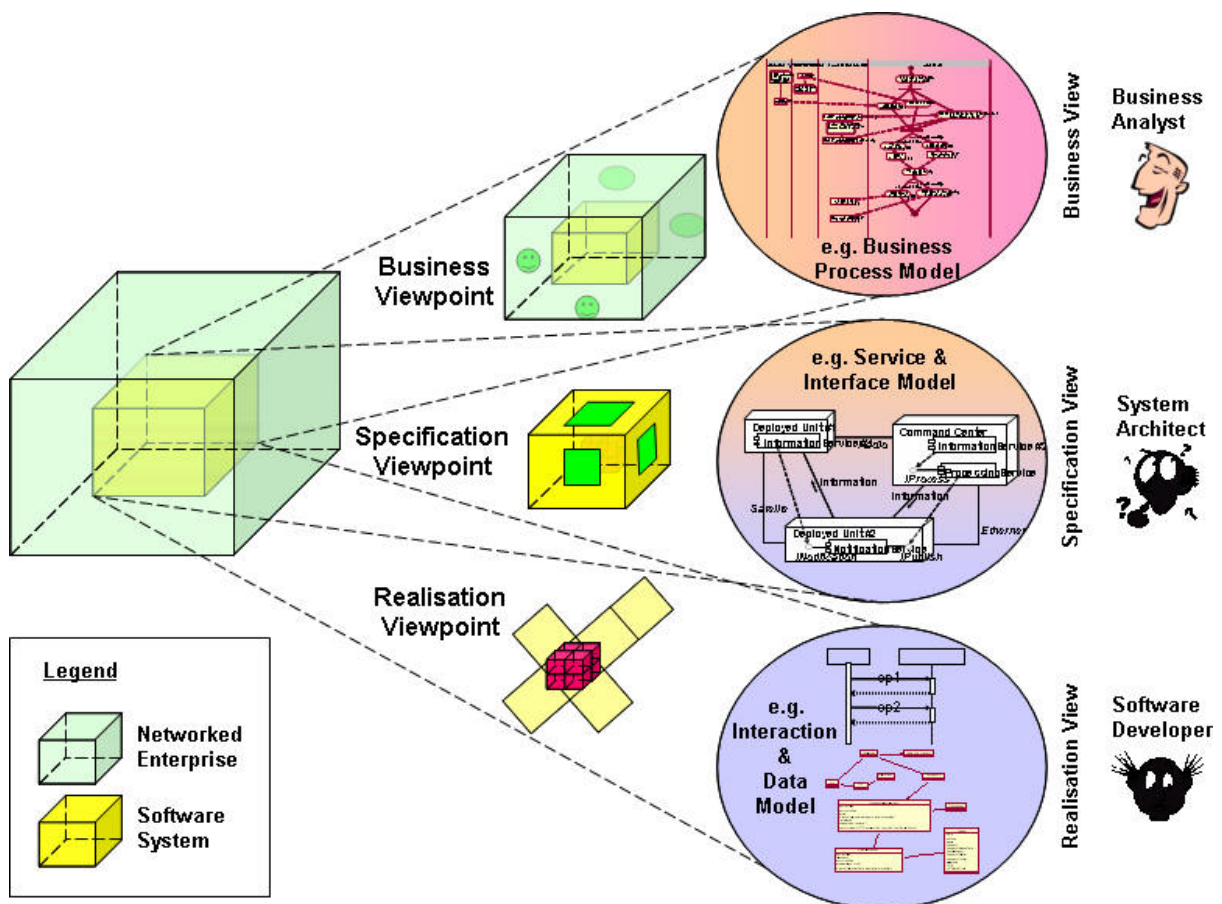
The MDD methodology needs to follow a structured approach where interoperability requirements from business operations in a networked enterprise drive the development of software solutions. This means that MDD methodology needs to be related to enterprise architectures. A specific part needs to address how the MDD concepts and the technical software components are reflected in a model world of the enterprise. The figure shows how the model world, reflecting the applicative integration, is related to the reference models for conceptual and technical integration. Enterprise and software models can be built to understand and analyse the interoperability requirements of an enterprise.

An enterprise model describes a set of enterprise aspects, which includes business models capturing actors and stakeholders, business services, business information and business processes. These business models provide a context for the software solutions that needs to

be developed and integrated.

Software models describe how software systems are used to support the businesses of an enterprise. The software models further refine the business models in terms of software realisation models. All these models should include descriptions of the four system aspects identified in the reference model for conceptual integration. The software models can be classified as CIM, PIM or PSM models according to a MDD abstraction.

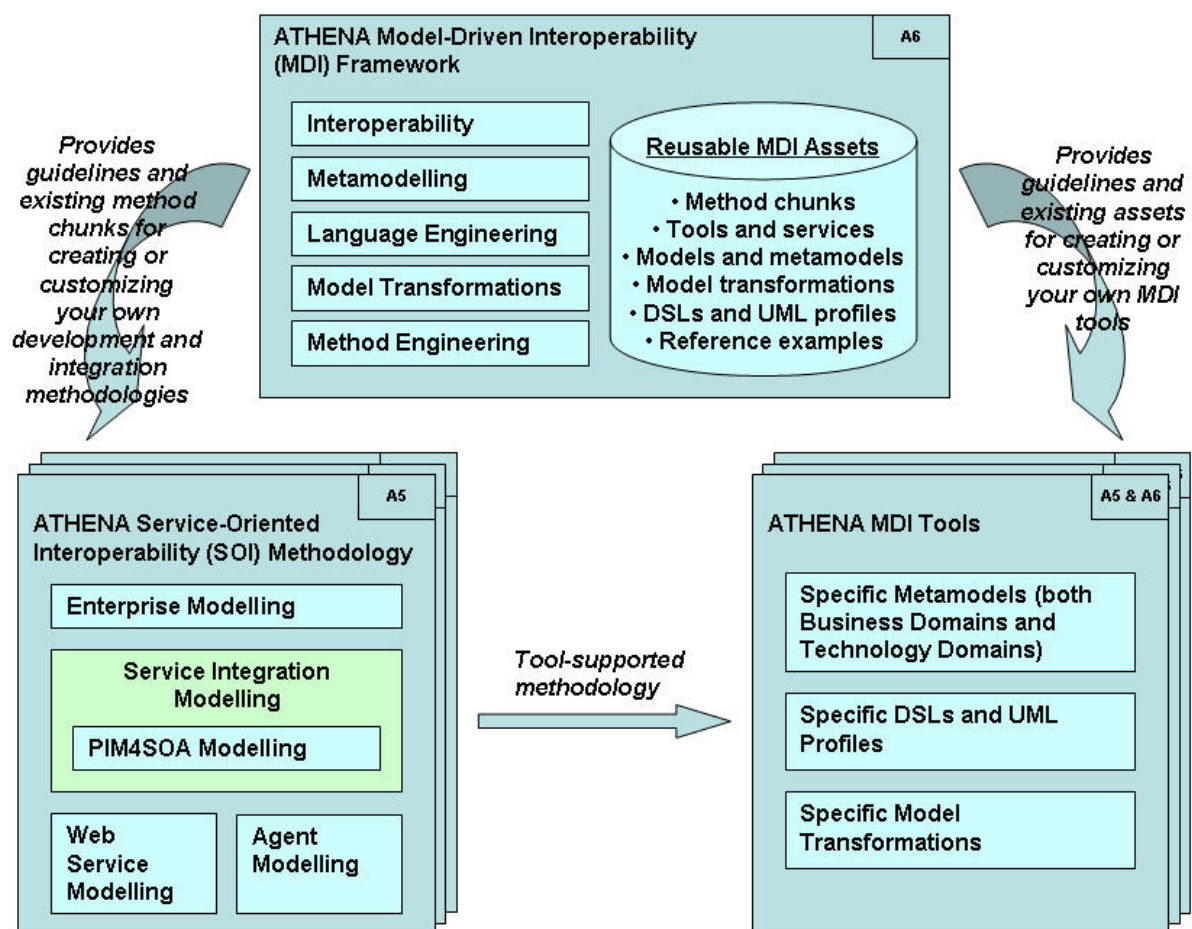
In our interoperability framework we have identified a set of models that we see useful in achieving interoperability. However, we also acknowledge the fact that this is just a baseline. Different enterprises must be able to develop their own software views that they see purposeful. It is important that the applicative integration supports the development of a set of shared views amongst different stakeholders and provides means for managing the dependencies between these views. We believe a viewpoint-based integration approach must be chosen. This allows incorporating viewpoints, which are implicitly or explicitly defined by other enterprise or software modelling approaches into an applicative framework.



We have identified three basic viewpoints that can be used as a starting point for viewpoint-based integration of software systems; *Business viewpoint* focusing on the

business context of the software system, *Specification viewpoint* focusing on the specification of the main components of the software system, and *Realisation viewpoint* focusing on the implementation of the software system. The figure illustrates the viewpoint metaphor exemplified using the three basic viewpoints. A business analyst is concerned with aspects related to the business context of the software system within the networked enterprise. A system architect is concerned with aspects related to the specification of the software system. A software developer is concerned with aspects related to the realisation of the software system.

3. Realisation of the framework



The reference models that we have just gone through specifies the conceptual part of the framework and identifies some of our MDD principles to interoperability. The realisation of the framework focuses on providing guidelines and existing method chunks for creating or customizing your own development and integration methodologies based on these principles, and guidelines and existing assets for creating or customizing your own MDI tool set.

The MDI framework aims at providing guidelines for the following topics:

- Model-driven architecture (MDA) and interoperability
- Metamodelling
- UML profiles and domain-specific languages (DSLs)
- Model transformations
- Method engineering

In addition it provides reusable assets in terms of method chunks, tools and services, models and metamodels, model transformations, DSLs and UML profiles and reference examples. The assets in the form of examples and tutorials will focus on applying Eclipse technologies for implementing your own MDI methodology and tool suite.

