Architecture design of the HORSE hybrid manufacturing process control system

Research School for Operations Management and Logistics

Architecture Design of the HORSE Hybrid Manufacturing Process Control System

P.W.P.J. Grefen, I.T.P. Vanderfeesten, G. Boultadakis (editors)

Beta Working Paper series 518

BETA publicatie	WP 518 (working paper)
ISBN	
ISSN	
NUR	
Eindhoven	November 2016

H2020 – FOF – 09 – 2015

Innovation Action

Smart integrated immersive and symbiotic human-robot collaboration system controlled by Internet of Things based dynamic manufacturing processes with emphasis on worker safety

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680734

D2.2a - Complete System Design - Public Version

Report Identifier:	D2.2a		
Work-package, Task:	WP2, T2.2	**Status – Version:**	Final - V1.0
Distribution Security:	Public	**Deliverable Type:**	
Editors:	Paul Grefen (Eindhoven University of Technology) Irene Vanderfeesten (Eindhoven University of Technology) George Boultadakis (European Dynamics)		
Contributors:	HORSE Architecture Team (HAT)		

Keywords:	System Design, Architecture
Project website: www.horse-project.eu	

Table of Contents

List of Figures

List of Tables

Abbreviations

BPMS	Business Process Management System
DoW	Description of Work
ERP	Enterprise Resource Planning System
EU	European Union
HAT	Horse Architecture Team
IEC	International Electrotechnical Commission
K4+1	Kruchten 4+1 Views Framework
MES	Manufacturing Execution System
MPMS	Manufacturing Process Management System
UML	Unified Modelling Language
UT5	Updated Truyens 5 Aspects Framework
WFM	Workflow management

Executive Summary

This document presents the complete system design of the HORSE project, concentrating on the logical architecture (as defined in the Kruchten 4+1 view framework) and providing a separation of concerns into software, data, process, organization and platform aspects (as defined in the Updated Truyens 5 aspect framework). The system design is based on the work in the HORSE requirements analysis task. Explicit confrontations with the HORSE system requirements specification are included.

This document starts with outlining the architectural approach to complete system design as used in the HORSE project and choosing the frameworks used in this approach. This is complemented with a choice of standards used. Next, we present the high-level HORSE scenario that specifies the functionality that the HORSE system should offer in an abstract, application context-independent way. This scenario specification follows from the HORSE system requirements specification. The functionality is next elaborated in a hierarchical logical software architecture, which in total contains five levels (including the context architecture for the HORSE system) This logical software architecture at the most detailed levels is the basis for software development for the HORSE system. The logical software architecture is complemented with data, process, organization and platform architectures.

HORSE
Complete System Design
Part 1:

Approach of
Complete System Design

1 Introduction

This section explains the purpose of this document (Section 1.1) and the structure of this document (Section 1.2).

1.1 Purpose of this document

In the Description of Work (DoW) of the HORSE project proposal, an illustration of the targeted HORSE system is included. This illustration is shown in Figure 1. The figure is a proper indication of where the project will be heading, but is not a proper specification that can serve as the basis for technology development. Put in informal terms, the figure can be considered an 'artist impression' of the projected HORSE system, but not an 'engineering blueprint'. The purpose of this document is to design and define such an engineering blueprint of the HORSE system that can be a solid basis for system development (and later deployment).

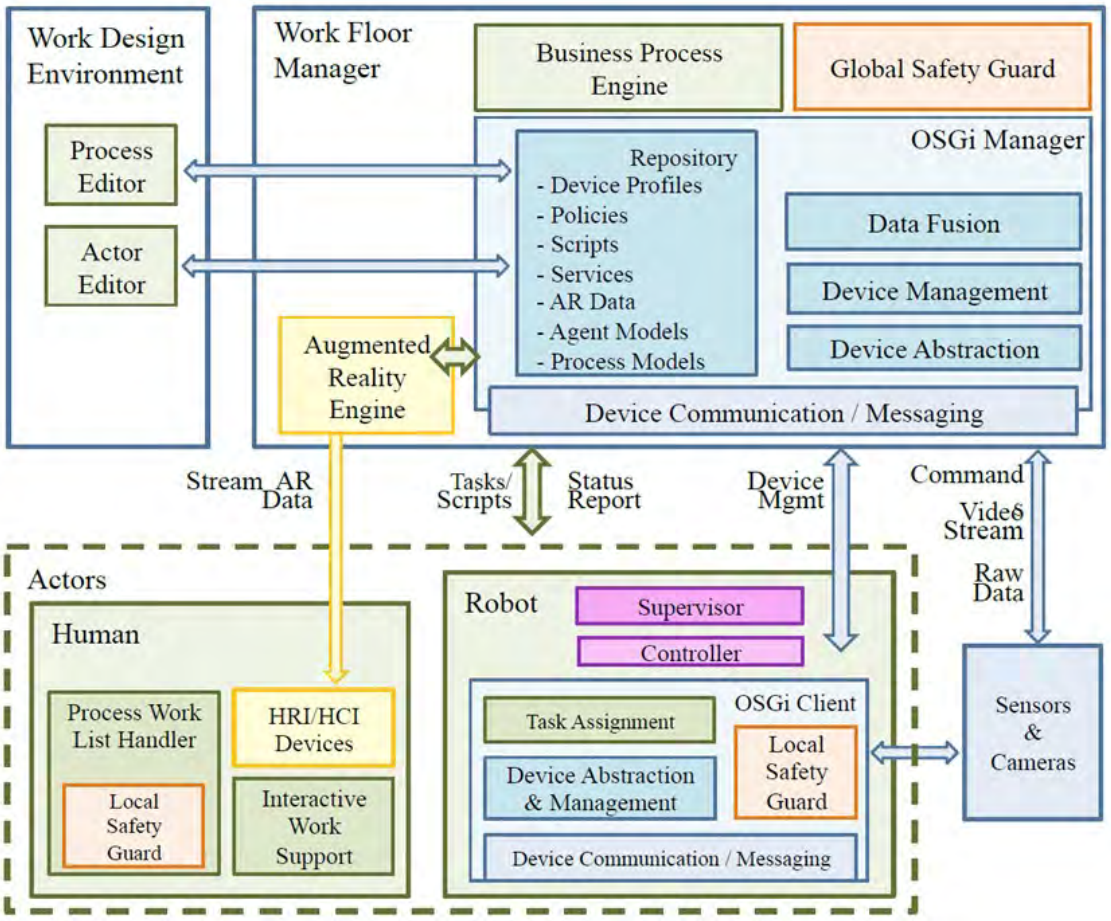

Figure 1: HORSE system illustration from DoW

The HORSE Architecture Team (HAT) is the body within the HORSE project responsible for designing the HORSE architecture and thus establishing the HORSE complete system design. The HAT takes architectural decisions with a project-wide scope.

1.2 Structure of this document

This document is organized in four parts, each containing a number of sections, as outlined below.

Part 1: Approach of complete system design

Section 1 contains this introduction.

Section 2 of this document explains the architectural approach to complete system design taken in the HORSE project, including the selection of two frameworks to achieve the required separation of concerns.

Section 3 explains how the work packages of the HORSE project are projected onto the K4+1 framework selected and explained in Section 2 to obtain a clear phasing in system development in the project.

Section 4 discusses the use of standards (both specification standards and manufacturing industry standards) in Complete System Design.

Part 2: High-level complete system design

Section 5 contains the abstract (i.e., not specific for a pilot case), high-level specification of the HORSE scenario, following from the HORSE requirements analysis [HOR16].

Section 6 contains levels 0-3 of the structural decomposition of the HORSE software architecture (following the UT5 aspect framework selected and explained in Section 2). The functionality specification resulting from the decomposition is checked at two levels against the top-down system requirements [HOR16].

Section 7 presents the HORSE logical data architecture by defining the high-level concept model. This concept model is required for the further decomposition of the logical software architecture to aggregation level 4.

Section 8 contains the design of the organization aspect of the HORSE logical architecture, explaining how the HORSE structure can be mapped to the organization of a manufacturing company.

Section 9 contains the design of the process aspect of the HORSE logical architecture. It discussed how enterprise-level business processes and manufacturing processes are related to HORSE concepts and structures.

Section 10 contains the design of the platform aspect of the HORSE logical architecture, i.e. the structure of the software and hardware required as the technical context for the HORSE system.

Part 3: Medium-level design of the software aspect

Sections 11 to 14 contain the logical software architectures at aggregation level 4 of the four main subsystems of the HORSE logical software architecture. These sections continue the design in Section 6, using the additional aspect architectures covered in Sections 7 to 10.

Section 11 covers the further elaboration of the HORSE Design Global subsystem at aggregation level 4.

Section 12 covers the further elaboration of the HORSE Exec Global subsystem at aggregation level 4.

Section 13 covers the further elaboration of the HORSE Config Local subsystem at aggregation level 4.

Section 14 covers the further elaboration of the HORSE Exec Local subsystem at aggregation levels 4 and 5.

Part 4: Conclusion, bibliography and appendices

Section 15 concludes the main body of this document.

The last sections of this document contain the bibliography and appendices that present additional details referenced in the main body of the document.

Appendix A contains a data flow analysis of the interface design of the logical software architecture at aggregation level 2.

Appendix B discusses a refinement of the database structure for the logical software architecture.

Appendix C contains a hierarchical overview of all software modules in the logical architecture with module identifiers.

2 Architectural approach to complete system design

In this section, we present the concepts and frameworks that form the basis for the approach that is taken in the HORSE project for complete system design. We take some space for this to ensure that all stakeholders involved in complete system design share the same background.

First we explain why we take architecture as a basis of this approach. Next, we discuss the architecture frameworks that have been selected by the AT: the Kruchten 4+1 View Framework (K4+1 for short) and the Updated Truyens 5 Aspect Framework (UT5 for short). Both frameworks have been chosen by the HORSE Architecture Team.

Given the complexity of the HORSE project and consequently of the HORSE system, a proper separation of concerns is required to be able to limit attention of the system design process in the right context to the right elements and aspects. In other words, separation of concerns avoids taking everything into consideration at every step of the way.

2.1 The role of architecture

An architecture of a software system or an information system can be seen as the high-level blueprint of that system that serves to understand its internal structure to aid in its design, redesign, configuration and maintenance – not so much different from the (technical) architecture of a complex building or a subway system. We now define the concept of architecture, starting with software architectures and then moving on to information system architectures.

2.1.1 Architecture of a software system

The concept of architecture of software systems originates from the domain of software engineering, i.e., the fields in which the detailed structuring of computer programs is the focus. The following is a definition of the term architecture from a well-known book in this domain [Shaw96]:

> *The architecture of a software system defines that system in terms of computational components and interactions between those components.*

The *computational components* form the *functional units* of a system, i.e., the parts that each provides something of the overall functionality. The term *computational* refers to the computations that a computer must make to realize the functionality. The *interactions* between the components allow them to operate as a whole to achieve the desired functionality of the software system.

The IEEE defines architecture as follows in its 1471 standard [Hill07], including the environment of a system and the design principles used to obtain and maintain the structure of a system:

> *Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.*

The relationships between the components and to the environment are the *connectors* or *interfaces*, which describe how the functional units interact (as stated in the first definition above) to achieve the overall behavior of the system, for example by exchanging information.

In HORSE, the environment is of great importance, as the HORSE system will interact with humans, robots and other systems that are not part of the HORSE system.

2.1.2 Architecture of an (enterprise) information system

The HORSE system can be seen as a complex software system in the software engineering tradition, but it can also be seen as an essential part of the enterprise information system of a manufacturing organization in the information system tradition. These two traditions are not mutually exclusive or contradictory, but put the emphasis on different points.

Viewing the HORSE system as part of an enterprise information system means that we don't look at small details of computer programs, but at larger structures of complex information systems. The complexity implies that we may have to focus on specific aspects of the system, that we have to distinguish between multiple levels and that we need structuring principles to understand the complexity (like the principles in the IEEE definition). Hence, we use the following definition of the term architecture in this document[1]:

> *The architecture of a (corporate) information system defines that system in terms of functional components and relations between those components, from the viewpoint of specific aspects of that system, possibly organized into multiple levels, and based on specific structuring principles.*

The concepts of *aspects* and *levels* are elaborated later in this document – here we will see that there are multiple aspects and multiple dimensions that can be used to define levels in an architecture specification. Depending on the chosen aspect(s), *functional components* can be of different natures: for example, they can be components that perform business functions (software components), components that hold business data (data storage components), or (parts of) business processes. Likewise, *relations* between components can be of different natures too: for example, they can be interfaces between software components or references between data storage components.

2.1.3 Architecture from a system point of view

From a system theory point of view, an architecture is a specification of a system. This system can be composed of *subsystems* and *aspect systems*.

Subsystems form a *structure decomposition* of a system: a set of sub-systems together form the super-system. Subsystems form a *partition* of a system: every element of the super-system is part of exactly one subsystem (this means that nothing is forgotten or replicated). To illustrate the concept, one can think of partitioning a house into multiple rooms: the house is the super-system formed by the combination of the rooms which are the subsystems. We will see later in this document that decomposition into sub-systems is a powerful way to deal with the complexity of the HORSE architecture. In designing a stepwise structural decomposition of an architecture, we traverse aggregation levels of that architecture [Gref15]: we move from a highly aggregated specification to a highly detailed specification.

Aspect systems form a *characteristics decomposition* of a system: a set of aspect systems together describe the structure and behavior of a system. The set of aspects provides a separation of concerns: we can look at each aspect of a system individually. To illustrate this concept, one can

[1] Note that the term 'architecture' is in general overloaded, as it can refer to a set of design principles (a set of norms or rules), to the process of applying these design principles (a process), and to the result of the application of design principles (a product). In our definition of the term 'architecture' and hence in the sequel of this document, we focus on the product interpretation, with proper attention for the relevance of the rules and process aspects. Other authors may take different positions, e.g. [Diet08] focuses on the rules interpretation.

think of the exterior design of a house versus the interior design: they are two aspects that describe characteristics of the same house, yet different aspects. We will see later in this document that working with aspects is also a powerful way to deal with the complexity of architectures.

Apart from structural decomposition and characteristics decomposition, we may also need several *levels of abstraction* [Gref15]. A highly *abstract architecture* is specified in general terms, such that it can be applied in multiple contexts (few context-specific parameters have been filled in, such as specific technology characteristics). A highly *concrete architecture* is specified in very specific terms, such that it matches one specific context (application scenario). In HORSE, we may need abstract architecture specifications that are applicable to multiple contexts, which are made specific to fit a specific pilot case.

2.2 The Kruchten 4+1 architecture framework (K4+1)

Kruchten has defined an architecture framework [Kruc95] that has become one of the most important standards in thinking about structuring software architectures. The main idea of the framework is to have a separation of concerns with respect to phases of the architecture specification and software realization process.

The framework organizes the description of an architecture around four main views with their respective main stakeholders:

1. The *logical view* specifies the object/module models of the design, i.e., the structure of the application logic in abstract terms. This view mainly specifies the functionality of a system under design, so *what the system should do*. The main stakeholders are the end users of the system.
2. The *development view* specifies the organization of the software in a development environment, i.e., the way the software development is supported to arrive at good software management. This view is concerned with getting good software, so *how the system should be realized*. The main stakeholders are the software engineers (programmers).
3. The *process view* specifies the concurrency and synchronization aspects of the software design, i.e., the way objects or modules in the logical view dynamically collaborate in parallel. This view mainly specifies the dynamic mechanisms and performance/scalability of a system under design, so *how it dynamically behaves*. The main stakeholders are the integrators of the system (those who connect modules).
4. The *physical view* describes the mapping(s) of software onto hardware, thereby reflecting the distribution aspect. This view mainly specifies the operational deployment of a system, so *what runs where*? The main stakeholders are the system engineers: those responsible for installing and maintaining the system.

As discussed above, each of the four views has its prime stake holders and its major concerns. This may lead to a content-wise divergence of ideas. To avoid this, the four basic views are illustrated by a fifth element:

5. The *scenarios* describe a few selected use cases that illustrate the four basic views. The scenarios make things concrete and provide a clear and practical basis for discussions between the various groups of stake holders (associated with the basic four views) in the architecture design or analysis. As such, the scenarios are the ‘content glue’ that provides convergence of ideas.

The five elements lead to the well-known Kruchten 4+1 model [Kruc95], shown in Figure 2. In this document, we abbreviate the name of this framework to *K4+1*.

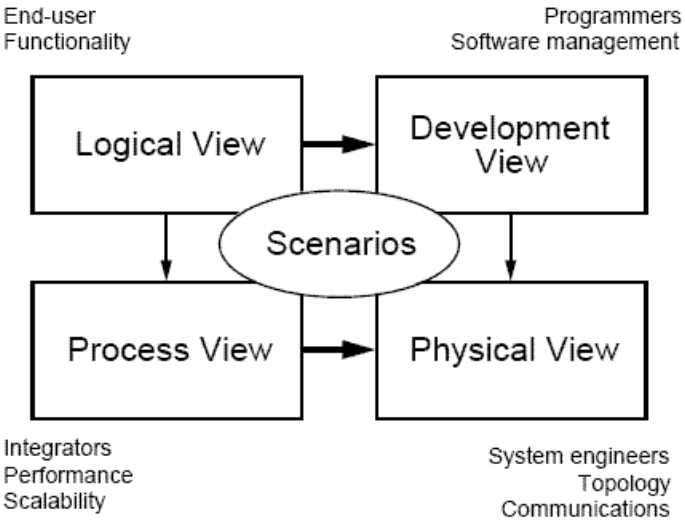

Figure 2: Kruchten 4+1 model (from [Kruc95])

2.3 The Updated Truyens 5 aspect framework (UT5)

The K4+1 framework provides a separation of concerns in terms of software development phases, but does not separate various aspects of the description of a complex software system or information system. For this purpose, we adopt an updated version of the 5 aspect framework of Truyens (abbreviated in this document as *UT5*). This framework was originally developed for information system development in the '90s [Truy90] and thereafter updated for information system developed in a modern, networked context [Gref15].

The UT5 framework is illustrated in Figure 3. It consists of five interconnected aspects, which we describe below.

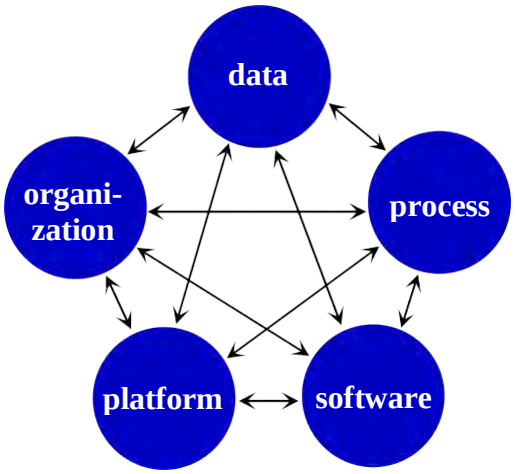

Figure 3: updated Truyens aspect framework (from [Gref15])

We distinguish between five interrelated architecture aspects in the design of the HORSE system:

Software: the software aspect describes the structure of the HORSE software under development; it is described for instance in UML component diagrams.

Process: the process aspect describes the structure of business/manufacturing processes that the HORSE system supports; it is described for instance in BPMN diagrams or UML activity diagrams.

Data: the data aspect describes the structure of data manipulated by the HORSE system, as well as the structure of the concepts that underlie data definitions (concept model); it is described for instance in UML class diagrams.

Organization: the organization aspect describes the structure of stakeholders in the HORSE context, such as operators of the HORSE system and designers of applications supported by the HORSE system; it is described by organigrams and/or actor models.

Platform: the platform aspect describes the structure of the existing technology that is required to run the HORSE system under design in its operating context; this includes both hardware (such as computer systems and robots) and software (such as existing enterprise information systems, middleware and hardware control software).

The five aspects provide a separation of concerns, but are interrelated as shown by the arrows in Figure 3. For example: if a data structure is changed (in the data aspect), it may be that the software that manipulates this data (in the software aspect) needs to be changed too.

3 The K4+1 framework applied to HORSE

In the HORSE project, several work packages (WPs) are devoted to system design, system realization, system deployment and system evaluation. To arrive at a well-structured overall development process, it is important that it is explicitly clear what is done in each work package - and likewise, what is not done in each work package. In other words, the development process needs to be clearly phased. For this purpose, the K4+1 framework is used as follows.

WP2 (Task 2.1) develops the *scenarios* of the K4+1 framework. We use three scenarios, which are based on the three HORSE pilot cases.

WP2 (Task 2.2) develops the *logical view* of the HORSE architecture, focusing on the end-user functionality of the system, its decomposition into functional modules and non-functional characteristics that are important to end users of the HORSE system. WP2 is also responsible for the hand-over of the logical view of the architecture to WP3 and WP4, including high-level decisions w.r.t. software development and software integration that explicitly follow from end-user requirements.

WP3 covers the *development view* of the HORSE architecture, focusing on the development of the software realization of the logical view. WP3 is also responsible for the hand-over of the development view to WP4, WP5, WP6 and WP8.

WP4 covers the *process view* of the HORSE architecture, focusing on integration of modules from WP3 and paying attention to performance and scalability constraints. WP4 is also responsible for the hand-over of the process view to WP5, WP6 and WP8. Note that this adds a dependency from the development view to the process view that is not part of the original K4+1 framework (which puts these two views in parallel).

WP5, WP6 and WP8 cover the physical view of the HORSE architecture, deploying the software developed in the development view using the integration developed in the process view. This will actually lead to running systems in the three pilots (WP5) and the selected open call cases (WP6 and WP8).

This leads to the situation illustrated in Figure 4. The additional dependency between WP3 and WP4 mentioned above is shown by the dotted arrow.

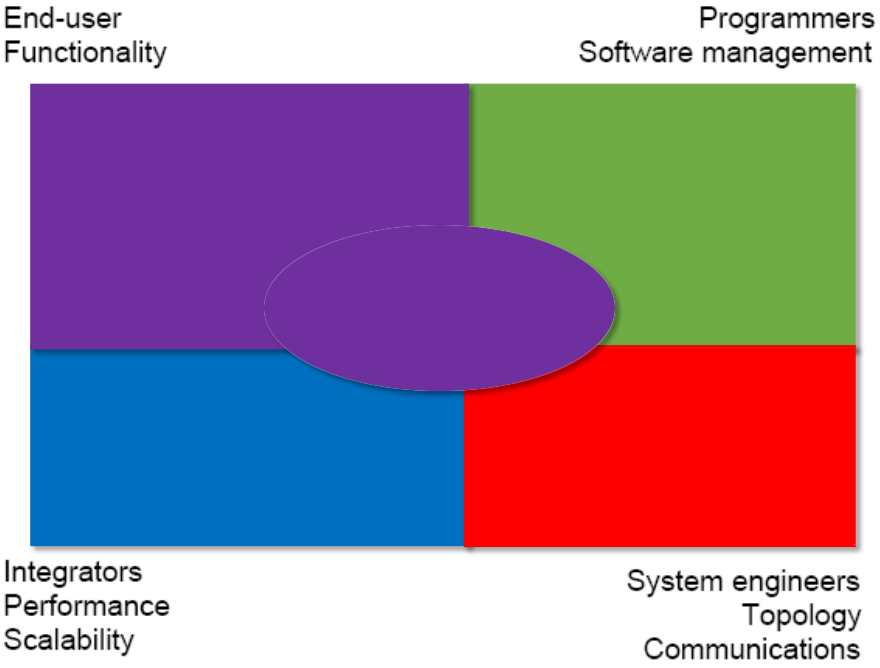

Figure 4: K4+1 related to HORSE work packages

WP3 results in the software of the HORSE system for physical deployment in WP5, WP6 and WP8. WP4 results in a description of how to deploy the software such that the right operational qualities (such as performance and scalability) are met in physical deployments. Thus, WP3 and WP4 together set the 'technical scene' for the other work packages.

4 Use of standards

In this section, we discuss the use of standards used in HORSE logical architecture design and specification. We cover both standards for architecture specification and application domain standards (i.e., manufacturing domain standards).

This section does not discuss technology standards used for embodiment of the designed architecture or software engineering standards used for realization of the HORSE software. These are part of the other architecture views according to the K4+1 framework, and hence will be dealt with in HORSE deliverables of other work packages - as discussed in Section 3.

4.1 Architecture specification techniques

It is important to choose a widely recognized standard for the specification techniques used in HORSE architecture design. The most used standard in software design is UML [Fowl03], which includes a number of important specification techniques. Hence, we adopt this standard in HORSE.

This includes the following specification techniques:

- For scenarios:
  - UML Use Case Diagrams.
- For software architectures:
  - UML Component Diagrams (static structure).
  - UML Sequence Diagrams (dynamic interaction).
- For data architectures:
  - UML Class Diagrams.
- For process architectures:
  - UML Activity Diagrams.

The choice for the UML standard should be pragmatic: we should not try to be perfect in adhering to technique standards, but use standards consciously such that they fit best. Using different standards than UML is permitted if good reasons for doing so exist (these should be explicitly stated).

4.2 Manufacturing domain standards

This standard is shown in Figure 5. In HORSE, we focus on discrete production, as indicated by the red dotted box in the figure.

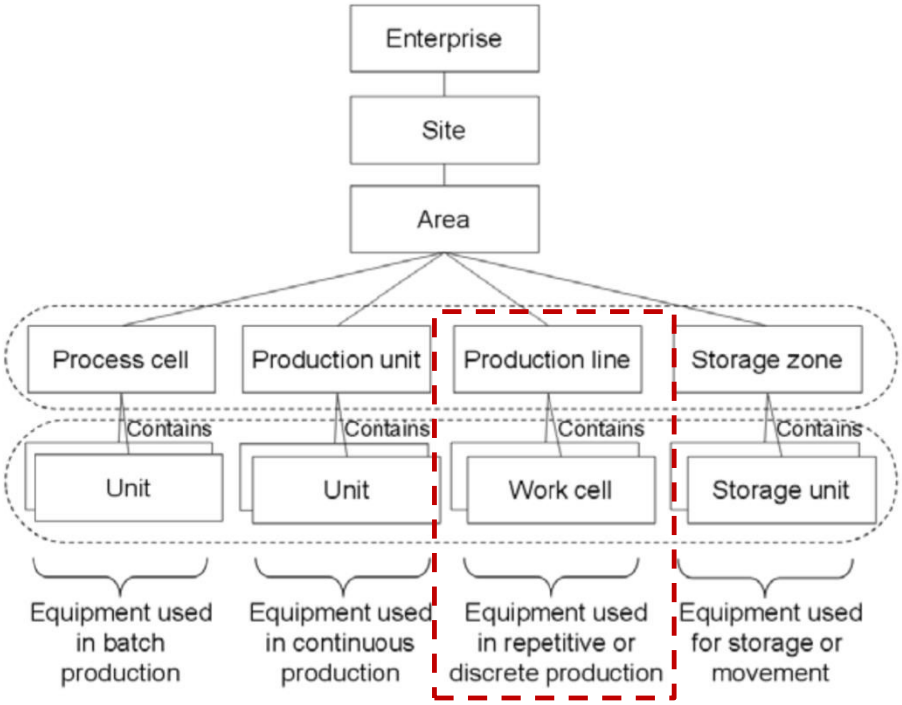

Figure 5: IEC manufacturing hierarchy standard [IEC13]

HORSE
Complete System Design
Part 2:

High-level Complete System Design

5 High-level HORSE scenario

In this section, we specify the high-level HORSE scenario (as in the K4+1 framework described in Section 2.2). This scenario is abstract, as it is not based on a specific HORSE pilot case, but on general requirements following from the HORSE requirements analysis [HOR16].

We use an abstract scenario to ensure that the HORSE system will be applicable to a wide range of application cases, including those of the three HORSE pilot cases and the open call cases projected in HORSE WP6 and WP8.

The scenario specifications can be seen as a kind of functional summary of the HORSE system requirements [HOR16].

5.1 Overall HORSE scenario

In Figure 6, we see the high-level use case view of the abstract HORSE scenario. The scenario is specified in a UML use case diagram [Heyw16]. A use case is a way in which a system can be used - it does not specify the internal working of a system.

This scenario includes four use cases:

- Design Manufacturing Process: functionality for the design of a manufacturing process across multiple work cells (and possibly across multiple production lines).
- Execute Manufacturing Process: functionality for the execution of a manufacturing process across multiple work cells (and possibly across multiple production lines).
- Configure Manufacturing Task: functionality for the configuration of a manufacturing task within a single work cell (possibly consisting of multiple manufacturing steps).
- Execute Manufacturing Task: functionality for the execution of a manufacturing task within a single work cell (possibly consisting of multiple manufacturing steps).

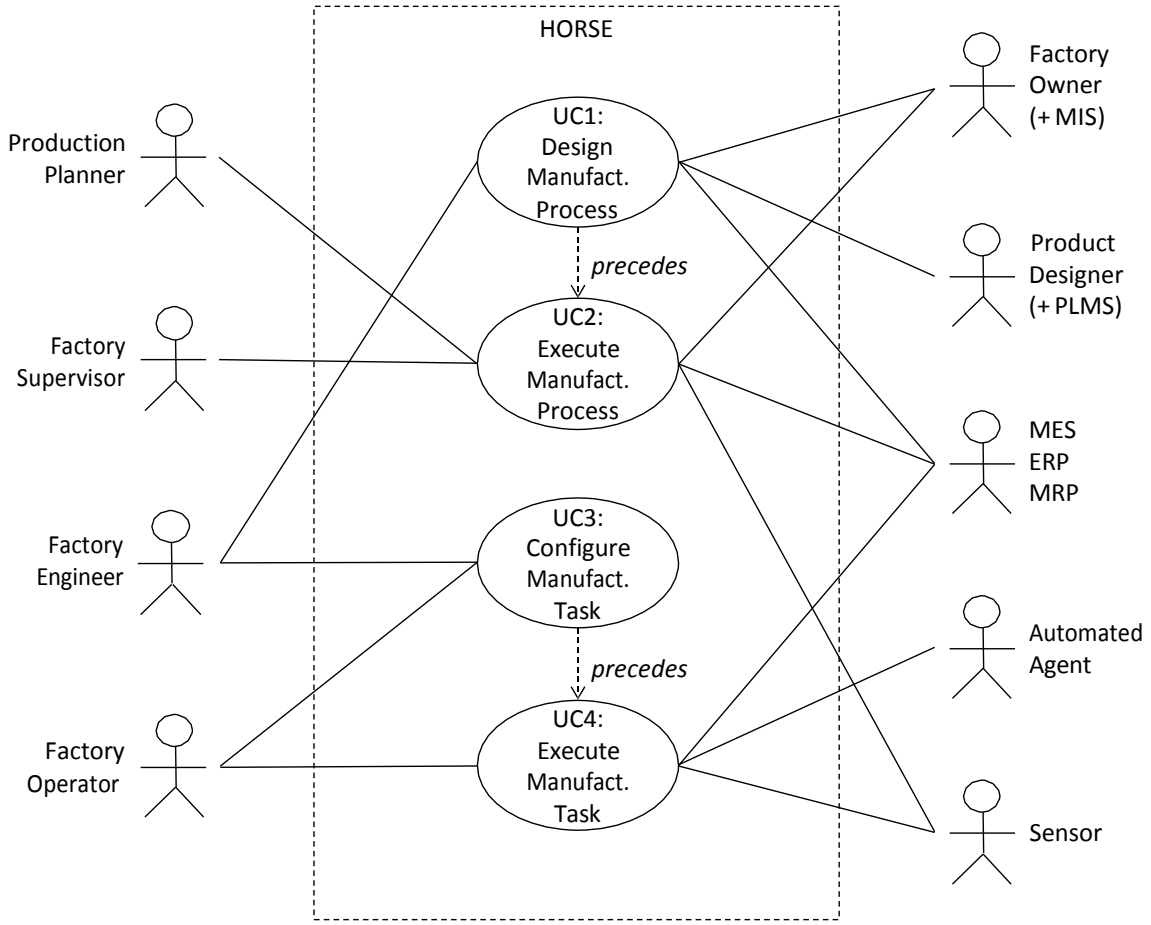

Figure 6: overall HORSE scenario

Each of the four use cases of Figure 6 can be further elaborated to show more detailed functionality. This is done in the next four subsections.

5.2 Use case Design Manufacturing Process

Figure 7 shows the elaboration of the Design Manufacturing Process use case of Figure 6. This sub-scenario shows the use of functionality for the design of a manufacturing process (at 'site' or 'area' level of the IEC hierarchy picture, as shown in Figure 5).

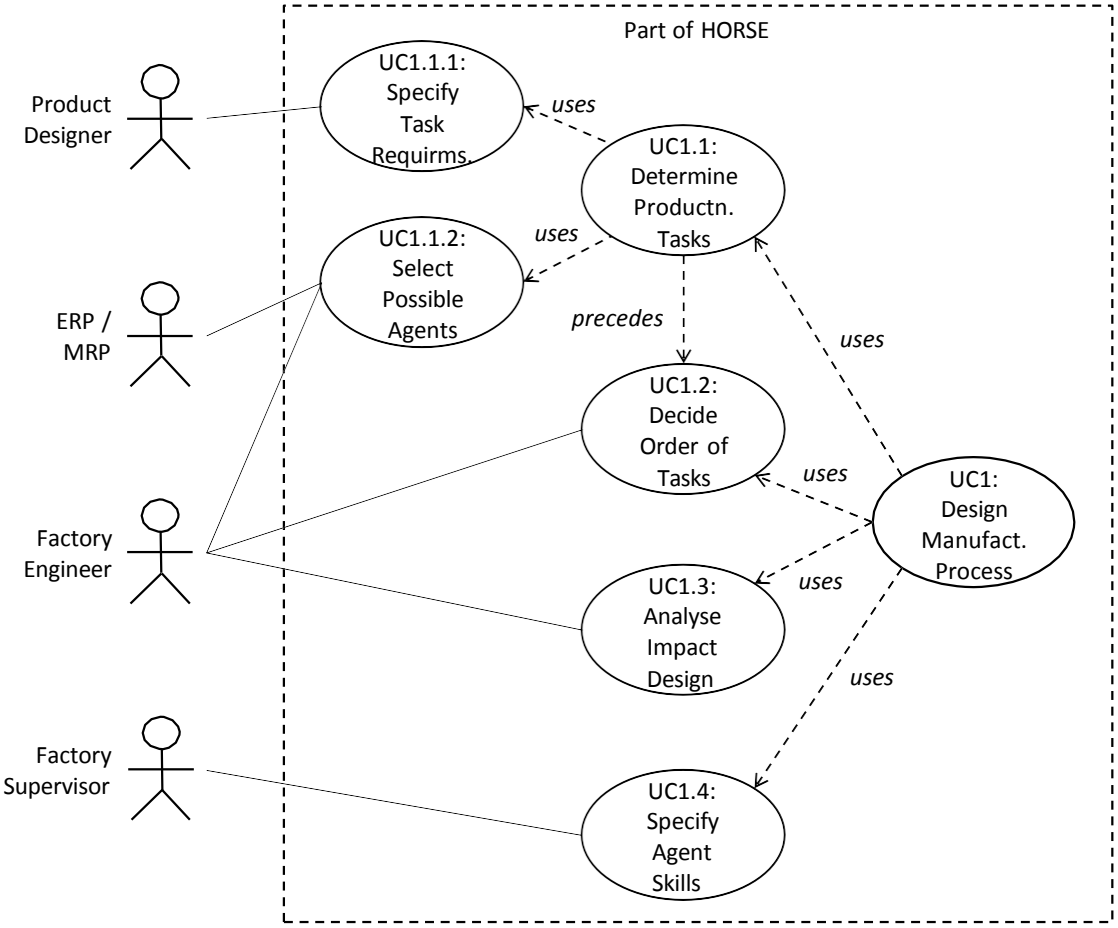

Figure 7: design manufacturing process scenario

5.3 Use case Execute Manufacturing Process

Figure 8 shows the elaboration of the Execute Manufacturing Process use case of Figure 6. This sub-scenario shows the use of functionality for the execution of a manufacturing process (at 'site' or 'area' level of the IEC hierarchy picture, as shown in Figure 5).

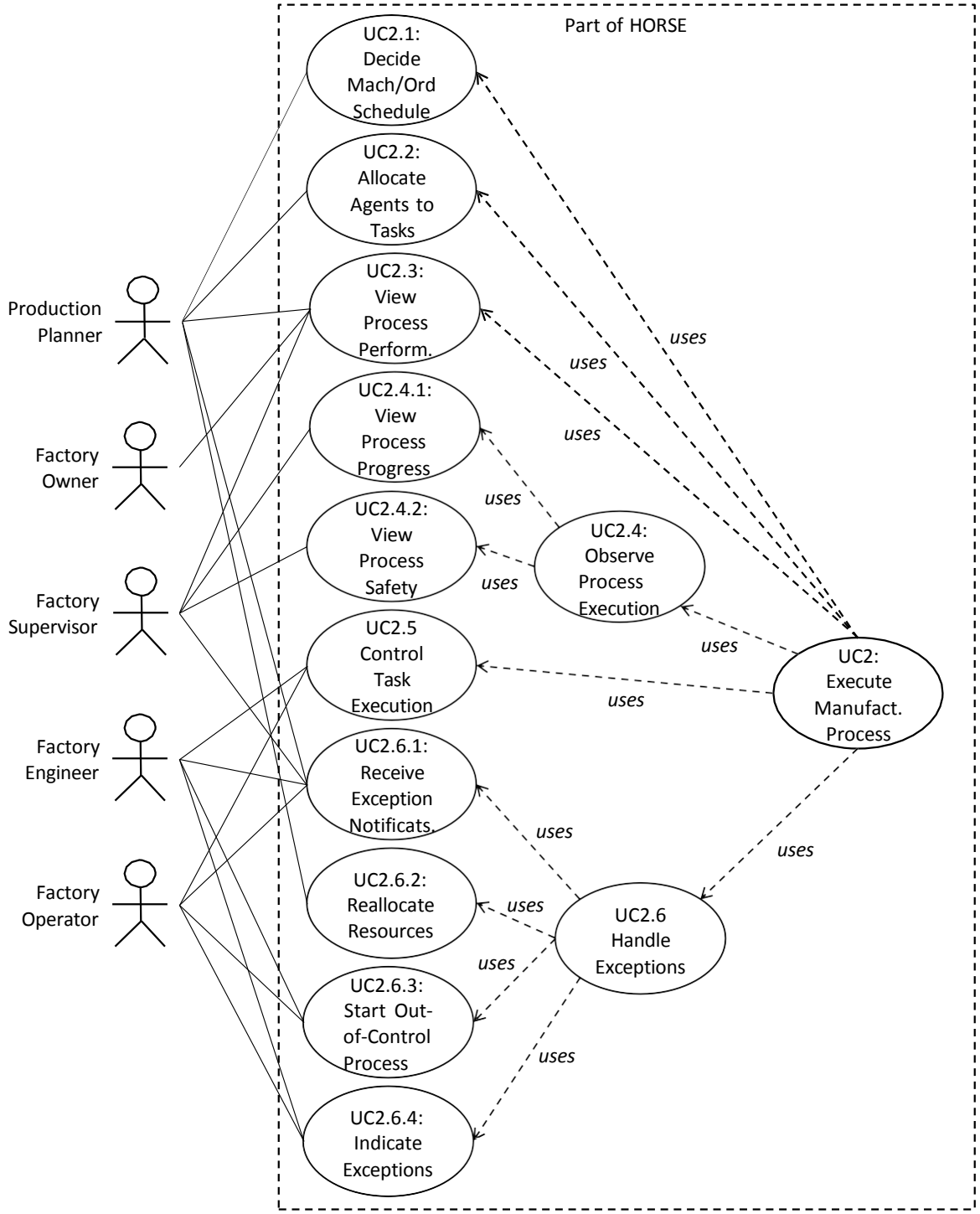

Figure 8: execute manufacturing process scenario

Note that there are no automated agents in the Execute Manufacturing Process scenario, as the automated agents are only active within the context of work cells, and hence only appear at the task level (see Section 5.5).

5.4 Use case Configure Manufacturing Task

Figure 9 contains the elaboration of the Configure Manufacturing Task use case of Figure 6. This sub-scenario specifies the use of functionality for the configuration of a single manufacturing task. This takes place in the context of a work cell (at the bottom level of the IEC hierarchy picture, as shown in Figure 5).

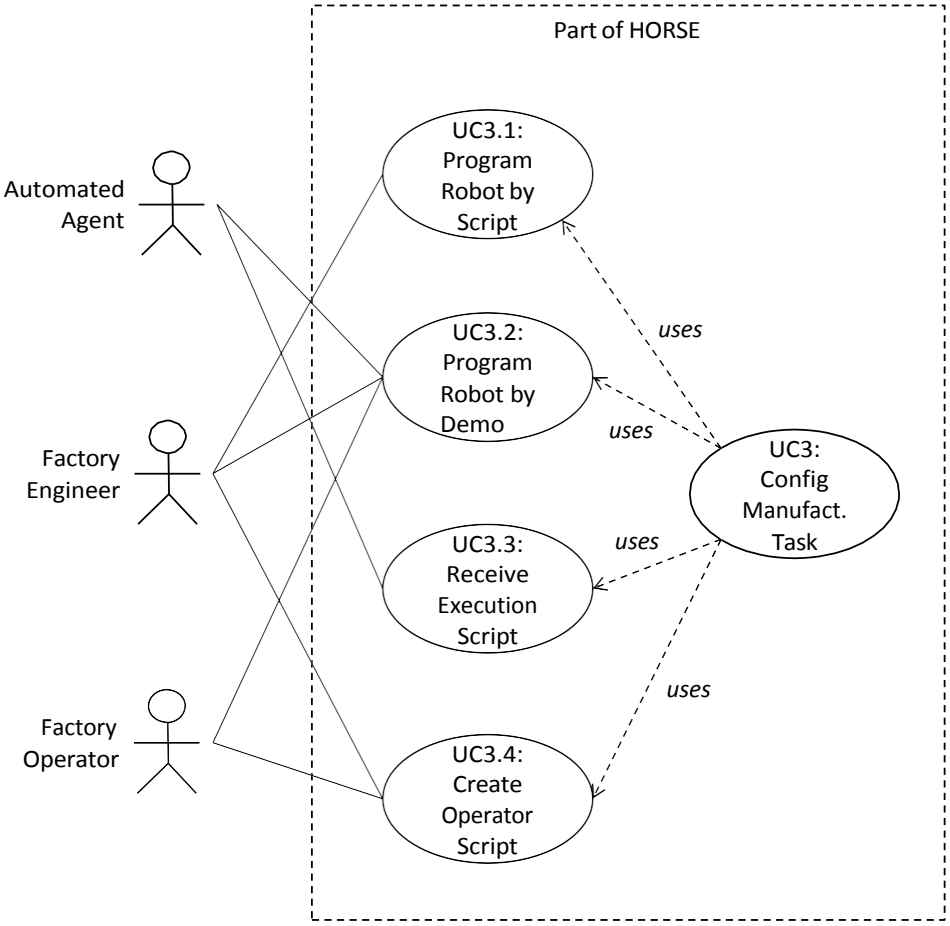

Figure 9: configure manufacturing task scenario

We place the following remarks with the Configure Manufacturing Tasks scenario:

- It is possible to configure a task in a hybrid way, i.e., use 'Program Robot by Script' and 'Program Robot by Demo' in a combined way.
- The use case 'Program Robot by Script' should cover graphical specification of an execution script for a robot.
- An execution script is a piece of data that is sent to a robot - it can result from various ways of specification (textual, graphical).

- We do not include a separate use case for testing/validating scripts. Physical testing/validating is seen as an iteration of configuration and execution of a task. Analytical testing/validating is seen as part of programming a script.

5.5 *Use case Execute Manufacturing Task*

Figure 10 shows the elaboration of the Execute Manufacturing Task use case of Figure 6. This sub-scenario specifies the use of functionality for the execution of a manufacturing step within a manufacturing work cell.

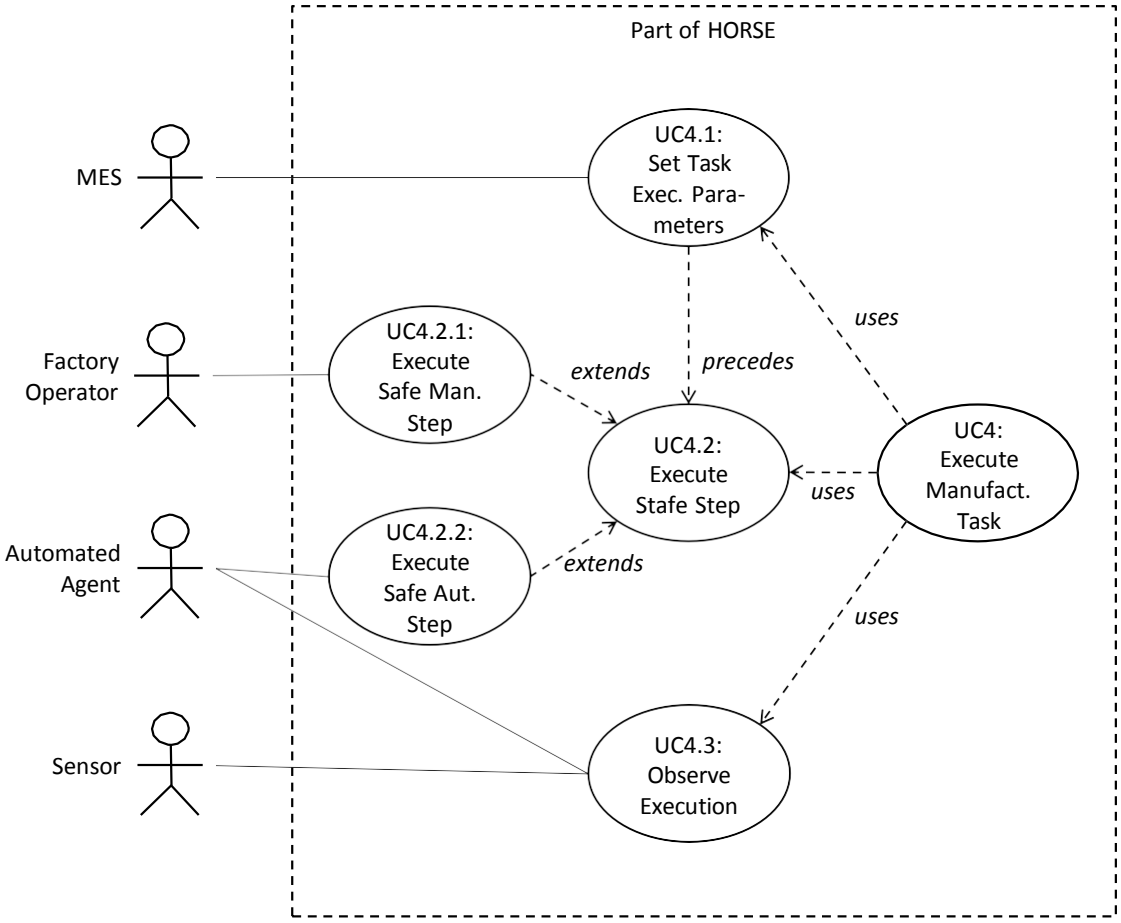

Figure 10: elaboration of Execute Manufacturing Task use case

We place the following remarks with the Execute Manufacturing Tasks scenario:

- We explicitly exclude the regulator as a role. He uses the same interface as the factory operator/factory engineer.
- The use case 'execute Safe Manual Step' includes Virtual Reality (VR) support. We don't include a specialization of the use case here as we don't aim at designing a safety certified VR/AR system.

6 Logical software architecture, aggregation levels 0-3

In this section, we start with the design of the logical architecture in the software aspect (as in the UT5 framework). We start with the software aspect, as this is leading in a system development project like HORSE. We follow a strict structural system decomposition approach (as explained in Section 2.1.3) to arrive at a well-defined architecture.

6.1 Logical software architecture, aggregation level 0

In the HORSE project, we develop a system for support of advanced manufacturing processes. These manufacturing processes take place as part of end-to-end business processes, i.e., processes starting at customer orders and ending in after-sales service. As such, various enterprise-level functions are linked, as modelled for instance in Porter's *value chain model* [Port85].

This means that the HORSE system will run in the context of other enterprise information systems supporting these processes, such as:

- enterprise-level business process management system (BPMS), which contains functionality to manage business processes across the various enterprise information systems;
- enterprise resource planning system (ERP), which contains functionality for the management/planning of customer orders, organizational resources, etc.;
- manufacturing execution system (MES), which contains functionality for the management of manufacturing resources;
- product lifecycle management system (PLMS), which contains functionality for the specification of products to be manufactured.

Consequently, it is a good idea to architecturally embed the HORSE system in an enterprise architecture context. As the exact enterprise information system context is dependent on a specific organization, we remain with a high-level, abstract software architecture here. This is shown in Figure 11.

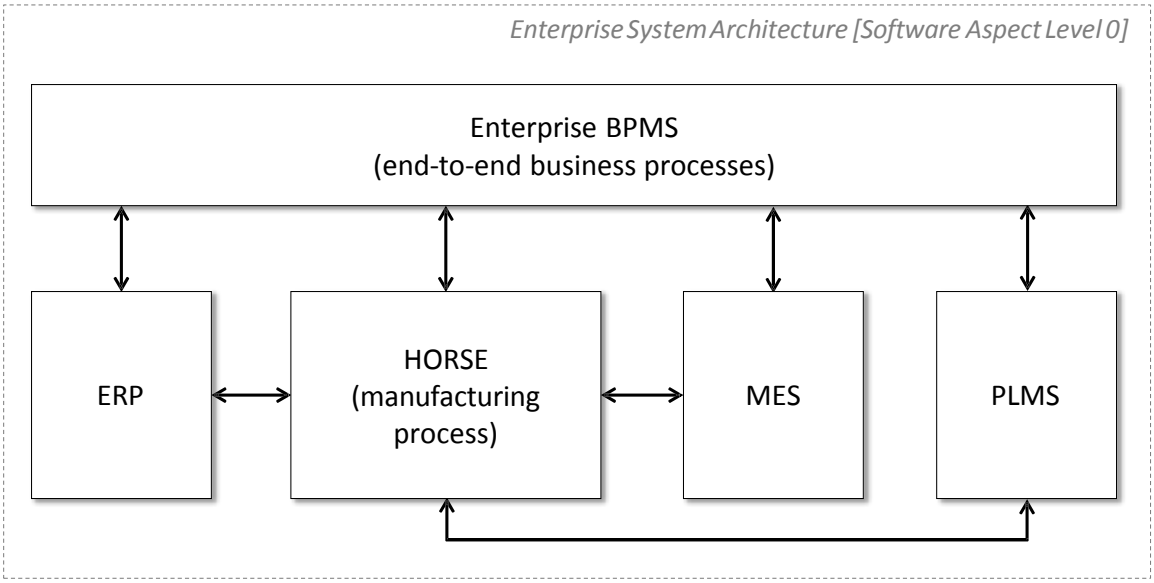

Figure 11: logical software architecture aggregation level 0

The Enterprise BPMS supports end-to-end business processes. This system is in the context of HORSE but not in the scope of the HORSE system design. The HORSE system supports the manufacturing processes. The relation between these two kinds of processes is further elaborated in the process aspect of the logical architecture, which is described in Section 9 of this document.

When mapped to the IEC manufacturing hierarchy standard [IEC13] (focused on discrete production), we get the situation shown in Figure 12.

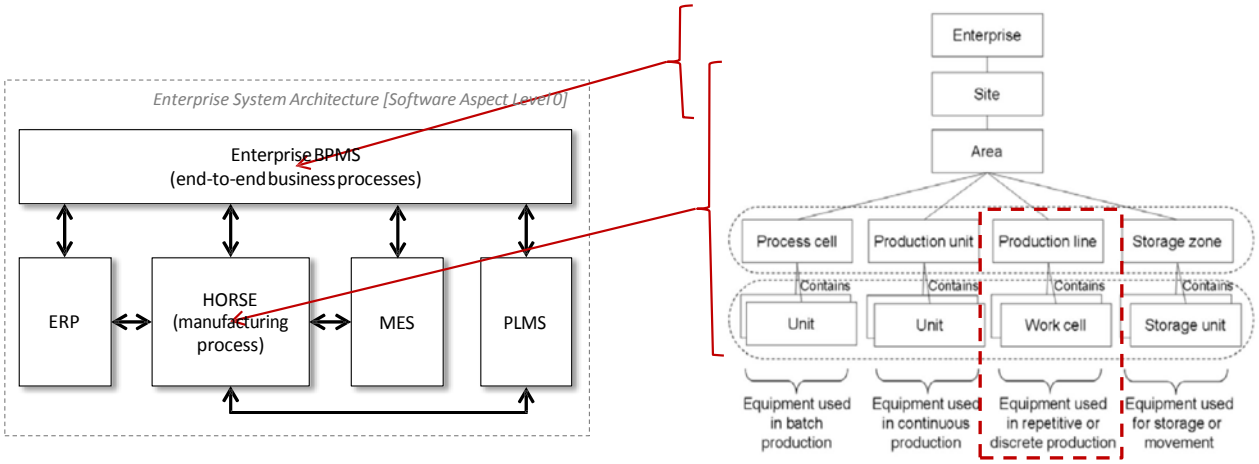

Figure 12: IEC manufacturing hierarchy mapped to HORSE software architecture level 0

End-to-end business processes can run at the site level (i.e., at a specific manufacturing site). Manufacturing processes can link multiple manufacturing areas, so they can be at the site level too. Consequently, these two kinds of process can both be at this level. In this situation, manufacturing processes (supported by the HORSE system) are part of end-to-end business processes (supported by the enterprise BPMS).

In the following, we concentrate on the HORSE system software architecture, refining (structurally decomposing) it into more detail.

6.2 Logical software architecture, aggregation level 1

The HORSE system must support both manufacturing processes across manufacturing cells and manufacturing steps within manufacturing cells. Typically, one manufacturing process uses a number of manufacturing cells - the process coordinates, the cells perform the actual work. This means that there are manufacturing activities at two distinct levels with different characteristics. Consequently, the HORSE software architecture has two levels as well, which we call *HORSE Global* and *HORSE Local*. We use these two levels to decompose the HORSE software architecture at aggregation level 0 (projected onto the HORSE system only) into the software architecture at aggregation level 1 - shown in Figure 13.

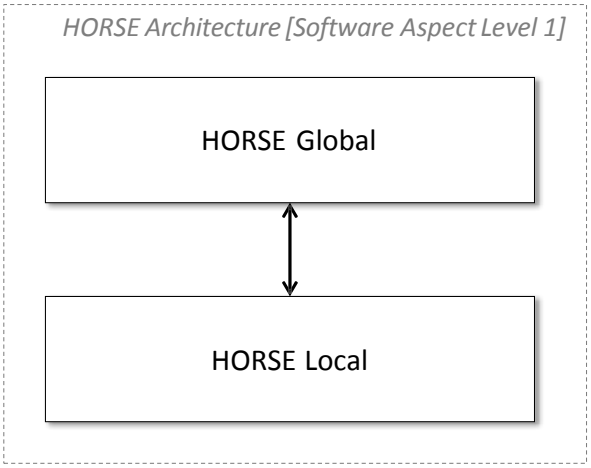

Figure 13: logical software architecture aggregation level 1

We can map the HORSE Global and HORSE Local levels again to the IEC manufacturing hierarchy [IEC13] (focused on discrete production). This is shown in Figure 14. Here we see that HORSE Global covers the site, area and production line levels of the hierarchy (as all these levels require coordination between work cells). HORSE Local covers the work cell level.

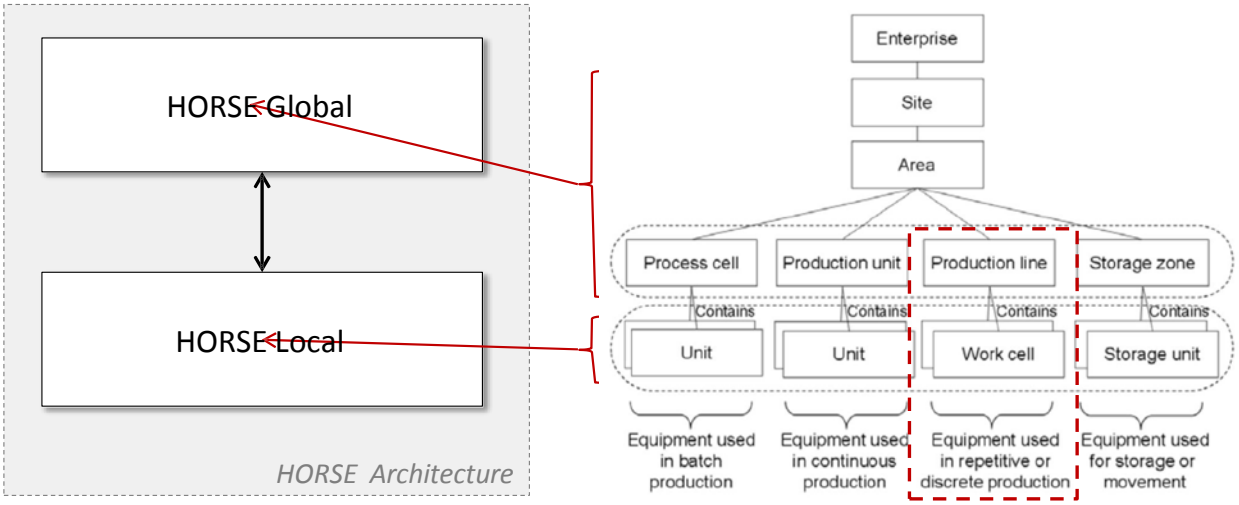

Figure 14: IEC manufacturing hierarchy mapped to HORSE Global and Local levels

6.3 Logical software architecture, aggregation level 2

In this section, we elaborate the logical software architecture at level 1, as shown in Figure 13. We first refine the architecture design. Then, we look at the interfaces. Finally, we check the resulting architecture against the HORSE system requirements [HOR16].

6.3.1 Architecture refinement

Manufacturing activities need to be designed or configured (to parameterize systems for specific production) and to be executed (to actually manufacture products). This means that the HORSE architecture needs to include modules for design/configuration and modules for execution. This holds both at the HORSE Global and HORSE Local levels.

This leads to a software architecture with four modules, as shown in Figure 15: two levels with each two modules. Note that we have not numbered modules in the logical software architecture diagrams in order not to clutter the diagrams. A hierarchical list of all modules in the logical software architecture with hierarchical module IDs is included in Appendix C of this document.

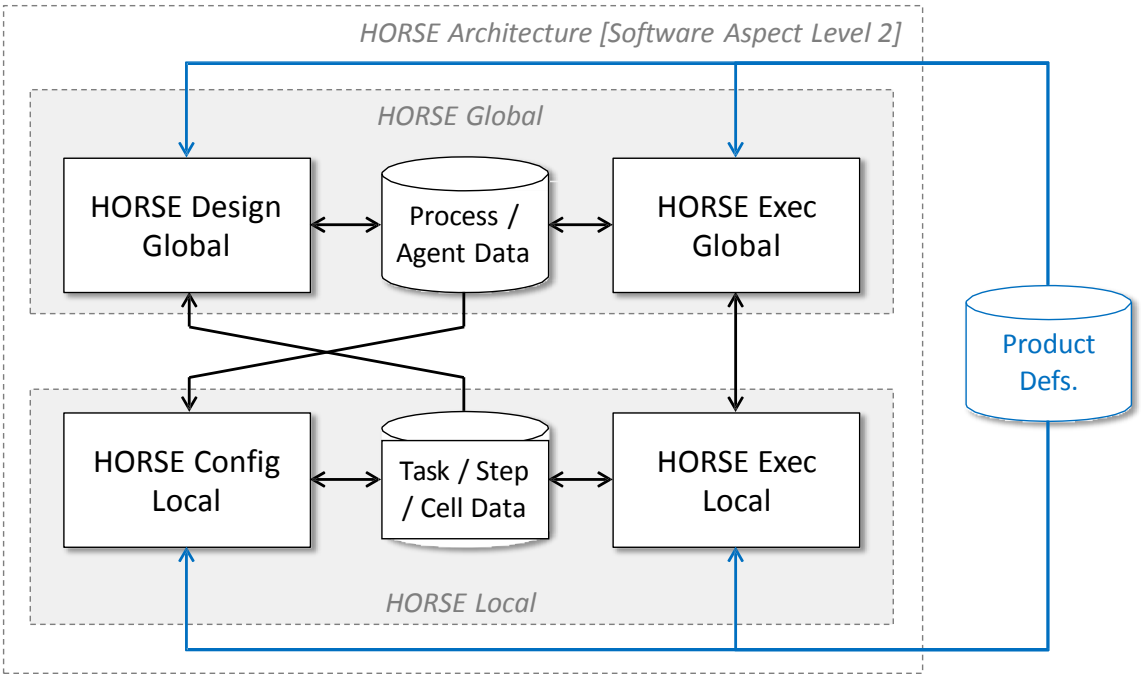

Figure 15: logical software architecture aggregation level 2

In the context of these four modules, we have placed a product definitions database (at the right hand side of Figure 15), as product characteristics partly determine design and execution of manufacturing processes, tasks and steps. This database is only read by the HORSE modules - it is maintained by the PLMS and/or ERP systems (as shown in Figure 11).

6.3.2 Database interface topology design

The design and execution modules at both the global and local levels are linked via databases at the respective levels - as shown in Figure 15. At the global level, we have a database containing definitions of manufacturing processes and manufacturing agents plus process execution data. At the local level, we have a database containing definitions of manufacturing tasks and steps plus execution data of tasks and steps.

The interface between design modules and databases is bi-directional, as the design modules read and write/update definitions. The interface between execution modules and databases is also bi-directional, as the execution modules read definitions and produce execution data.

Note that we keep the definition databases and execution databases combined in the architecture diagrams (also in the further refinement in this document) to not clutter these diagrams. Many modules use both databases, so separating them in the diagrams would lead to many additional interfaces. To illustrate this point, we show the architecture of Figure 15 with separated definition and execution databases in Figure 16.

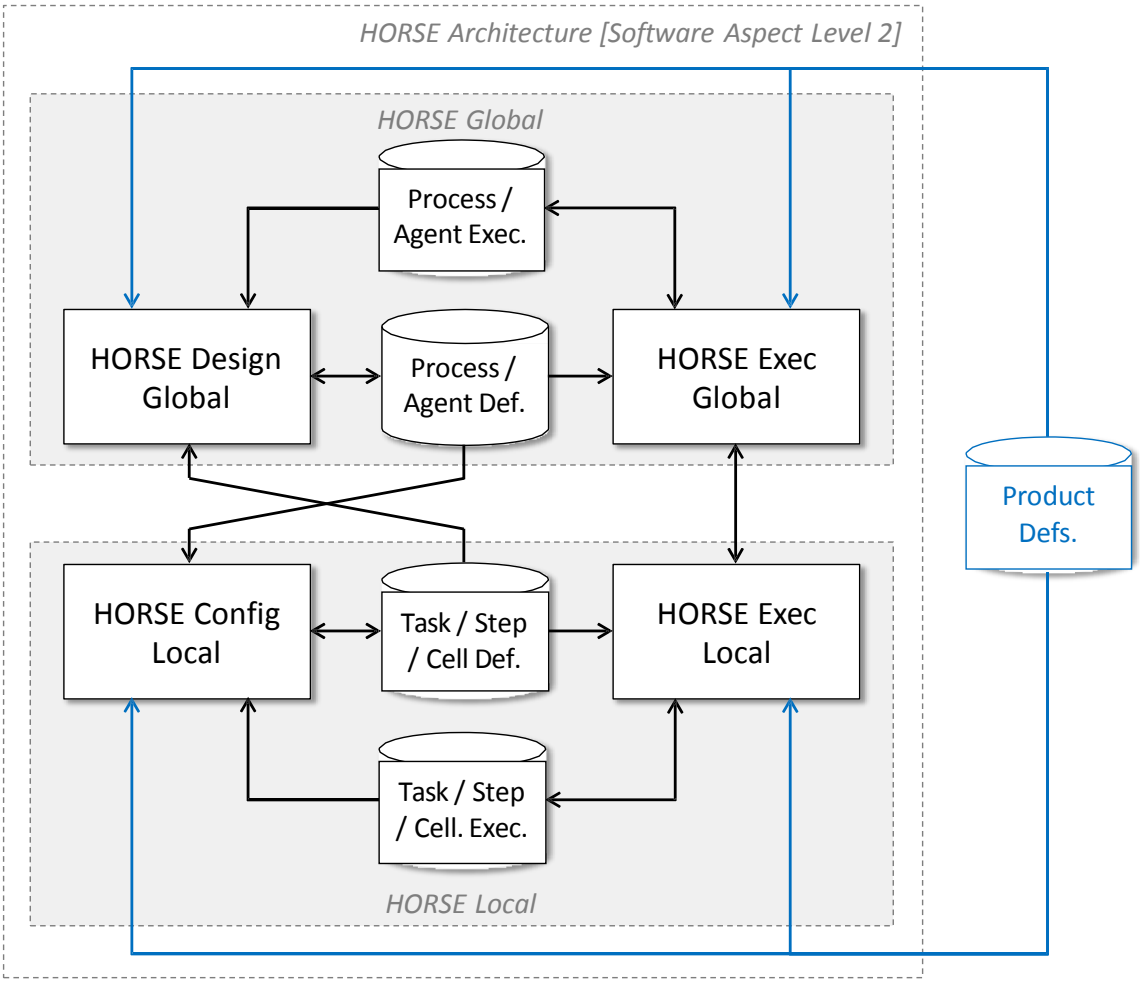

Figure 16: logical software architecture aggregation level 2 with separated databases

The same transformation can be performed for more refined architectures that follow in this document (with greater complexity because of greater numbers of software modules) - we omit them for reasons of clarity and brevity. Note also that the databases can even be further split into more specific databases (as defined for example in the Mercurius reference architecture for workflow management systems [Gref98]). A possible logical refinement of databases is discussed in Appendix B of this document. We leave a final design as an implementation issue for the development view of the architecture.

On the design/configuration side, the two levels are linked. The global process definitions are also input to the HORSE Config Local module, as process definitions form the context of task definitions. Likewise, the local task definitions are input to the HORSE Design Global module, as task definitions are 'sequenced' in process definitions. This leads to the inter-level database interfaces in Figure 15.

6.3.3 Execution interface topology design

On the execution side, the two levels are linked by a bi-directional connection between both execution modules. Downwards, this link is used to communicate execution commands. Upwards, this link is used to communicate execution statuses. We take the explicit decision that there is one

generic interface between HORSE Global and HORSE Local. This interface covers all type of work cells: human-only work cells, fully robotic work cells, hybrid work cells and even work cells that consist of sensors only.

In a practical situation, the HORSE Exec Local module is replicated (one logical module instance per work cell as in Figure 14). This is illustrated in Figure 17 for a situation of a factory with three manufacturing cells. In the logical software architecture, all communication between HORSE Exec Local module instances takes place through the HORSE Exec Global module, i.e., logically HORSE uses a centralized orchestration paradigm to coordinate manufacturing cells.

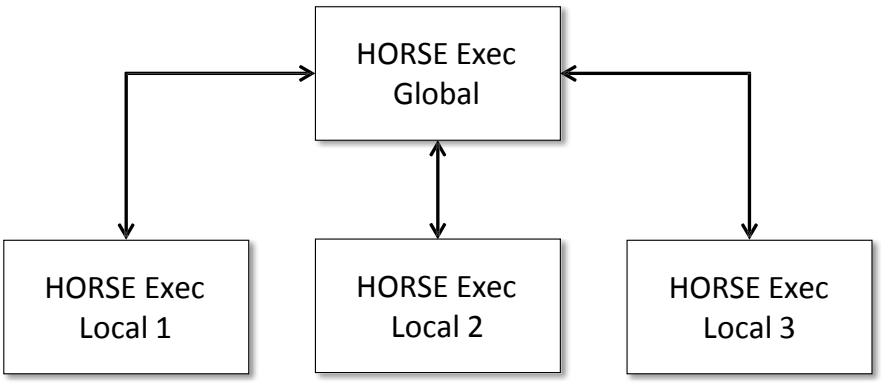

Figure 17: HORSE Exec Global and HORSE Exec Local

In the development/process architecture views, timing and availability constraints in real-time situations (i.e., non-functional requirements) may give rise to the necessity of a direct peer-to-peer connection between instances of HORSE Exec local modules. This is captured in an architecture advice in the hand-over to WP3 and WP4 (see Figure 4).

6.3.4 Interface data structure and message design

In Figure 18, we repeat the logical software architecture aggregation at level 2 of Figure 15, without the product database, but with interface IDs between the HORSE modules.

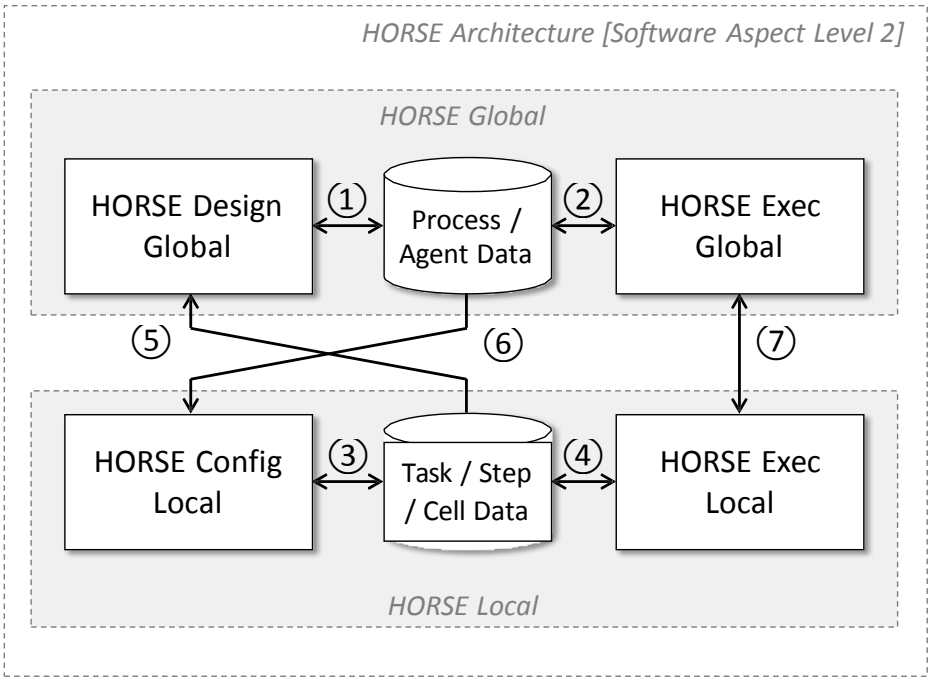

Figure 18: logical software architecture aggregation level 2 with interface IDs

The interfaces shown in Figure 18 carry data structures (possibly in messages) between subsystems. These data structures need to be specified.

Interface	Direction	Data	Remarks
①	HDG → PD	Process models (sequencing of tasks) Agent models (incl. capabilities) Allocation models (role models)	
	PD → HDG	Process models (sequencing of tasks) Agent models (incl. capabilities) Allocation models (role models) Process performance data	
②	PD → HEG	Process models (sequencing of tasks) Agent models (incl. capabilities) Allocation models (role models) Process performance data (exec. log)	
	HEG → PD	Process performance data	
③	HCL → TSD	Task and step model definitions	
	TSD → HCL	Task and step model definitions Task and step performance data	
④	TSD → HEL	Contents of task (work instructions / scripts)	
	HEL → TSD	Task and step performance data / statistics	
⑤	TSD → HDG	Black box characteristics of task definitions	
⑥	PD → HCL	Capability models (incl. capabilities)	
⑦	HEG → HEL	Task control commands (see *additional remarks*) Product definitions (references)	Synchronous interface
	HEL → HEG	Alerts Measurements Task control confirmations Task statuses	

Table 1: interfaces in logical software architecture aggregation level 2

Additional remarks with respect to interfaces in the logical software architecture aggregation at level 2:

- Task control commands passed from HEG to HEL do not include work lists (to-do lists for agents) for now, since manufacturing shop floor does not seem to leave enough freedom/autonomy for human agents.

To check the completeness and consistency of the interface topology listing in Table 1, a data flow analysis has been performed. This analysis is presented in Appendix A of this document.

6.3.5 Confrontation with requirements

To check the logical software architecture at level 2 (as shown in Figure 15), we confront it with the set of top-down functional requirements at level 2 [HOR16]. The purpose of this confrontation is to check:

- whether each functional requirement is covered by at least one module in the architecture (completeness);
- whether each architectural module is required by at least functional requirement (minimality).

The result of the confrontation is shown in Table 2, with the requirements in the rows and the architectural modules in the columns. As all rows and columns are filled, the confrontation has a positive outcome. We repeat this analysis in a refined form at aggregation 3 of the logical software architecture (see Section 6.4.6).

	HORSE Design Global	HORSE Exec Global	HORSE Config Local	HORSE Exec Local
AF-01: Situation awareness		X		X
AF-02: Synchronization of robotics and human activities	X	X	X	X
AF-03: Robotics task instructions			X	X
AF-04: Human task instructions			X	X
PF-01: Horizontal Business Process Management	X	X		
PF-02: Resource management	X	X		
PF-03: Actor Control		X		X

Table 2: confrontation with top-down requirements Level 2

6.4 Logical software architecture, aggregation level 3

At aggregation level 3 of the logical software architecture, we elaborate each of the four functional modules identified at level 2 (see Figure 15).

6.4.1 HORSE Design Global

The HORSE Design Global subsystem contains functionality to design manufacturing activities at the global level, i.e., at the site, area and production line levels (see Figure 14). This design involves two aspects:

- design of the manufacturing processes, i.e., what needs to happen in which order and with what requirements to the agents involved (role specifications);
- design of manufacturing agents, i.e., the humans and machines (robots and other relevant automated machines) with their characteristics.

The two aspects are mapped to two logical software modules at this aggregation level. These software modules interact through the Process Definitions database (i.e., this subsystem has a data-centered architecture).

The resulting architecture is shown in Figure 19. The architecture also shows the contextual connections following from the software architecture at Level 2 (see Figure 15).

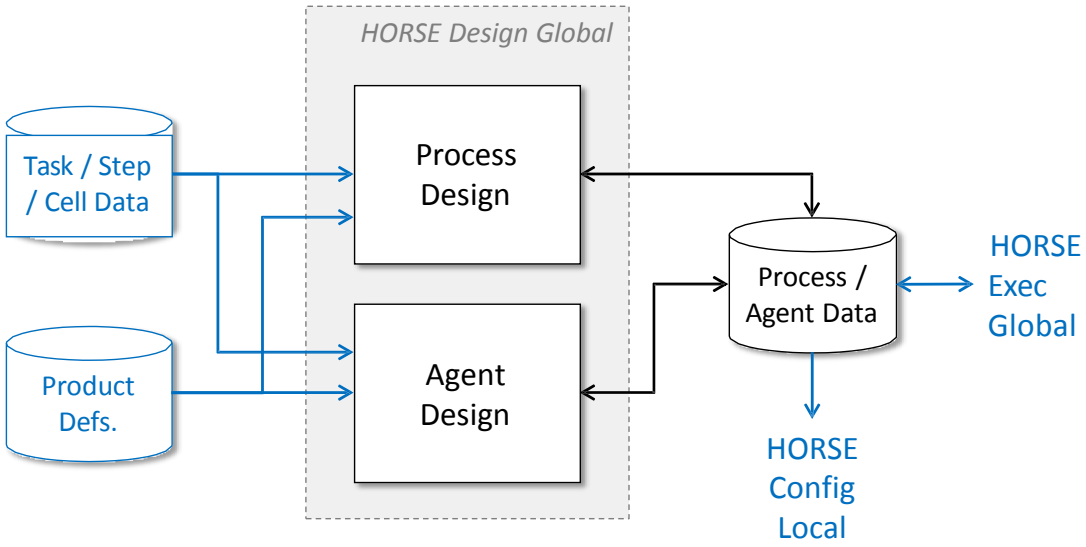

Figure 19: logical software architecture aggregation level 3, subsystem 1

6.4.2 HORSE Exec Global

The HORSE Exec Global subsystem is responsible for manufacturing activities across work cells, i.e., at the site, area and production line levels (see Figure 14). This involves two main functions:

- supporting execution of manufacturing processes, i.e., making things happen;
- supporting awareness about the global state of execution, i.e., observe things that happen and processing this into relevant signals for controlling execution.

Both main functions are allocated to logical software modules at this aggregation level. We label the execution module Manufacturing Process Management System (MPMS), as a variation of a standard Business Process Management System (BPMS, mostly applied in the administrative domain).

It is important that design decisions w.r.t. the internals of the HORSE Exec Global module (either in the logical architecture or later in the development and process architectures) are as much as possible isolated from design decisions in the HORSE Exec Local module. For this reason, we include an abstraction layer (Exec Global Abstraction layer) in the interface to the HORSE Exec Local subsystem.

The above decisions lead to the logical HORSE Exec Global architecture shown in Figure 20. The architecture also shows the contextual connections following from the software architecture at Level 2 (see Figure 15).

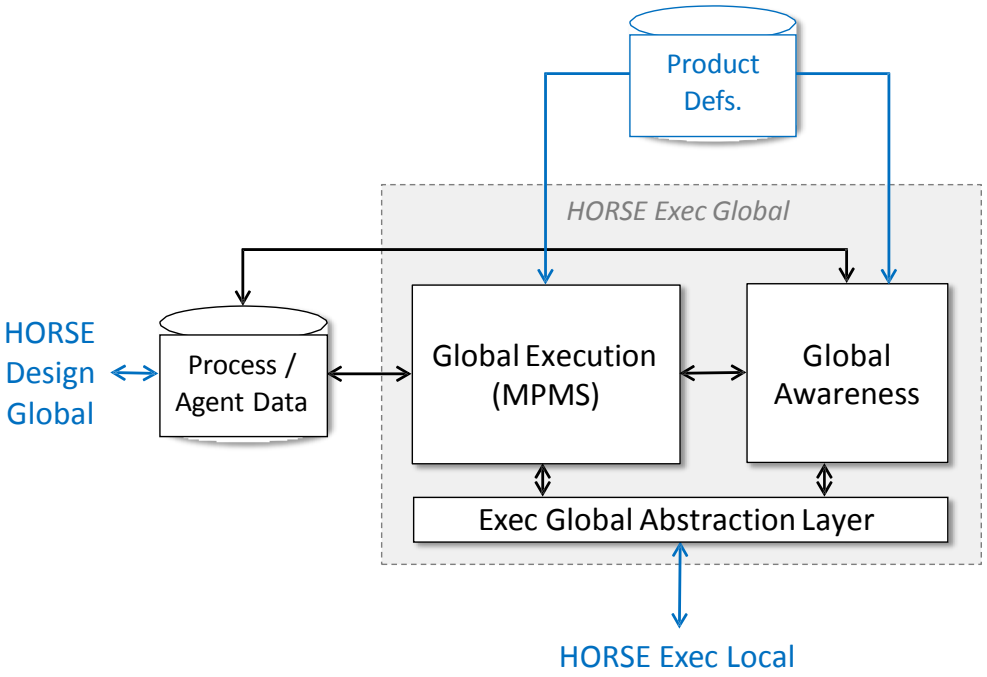

Figure 20: logical software architecture aggregation level 3, subsystem 2

6.4.3 HORSE Config Local

The HORSE Config Local subsystem contains functionality to design manufacturing activities at the local level, i.e., at the work cell level (see Figure 14). This design involves three main aspects:

- configuration of manufacturing tasks, i.e., the high-level activity spanning a work cell; note that this may require multiple agents of different kinds that each execute manufacturing steps (e.g. a human and a cobot);
- configuration of manufacturing steps, i.e., the low-level procedures performed by humans and machines (robots and other relevant automated machines);
- design of workcells.

The three aspects are mapped to four logical software modules at this aggregation level (by distinguishing between human step design and automated step design, as these require different functionalities). These software modules interact through the Task/Step/Cell Definitions database (i.e., this subsystem has a data-centered architecture).

The resulting architecture is shown in Figure 21. The architecture also shows the contextual connections following from the software architecture at Level 2 (see Figure 15). Note that the Process Definitions database is only connected to the Task Design module, as tasks are embedded in processes (and steps are embedded in tasks).

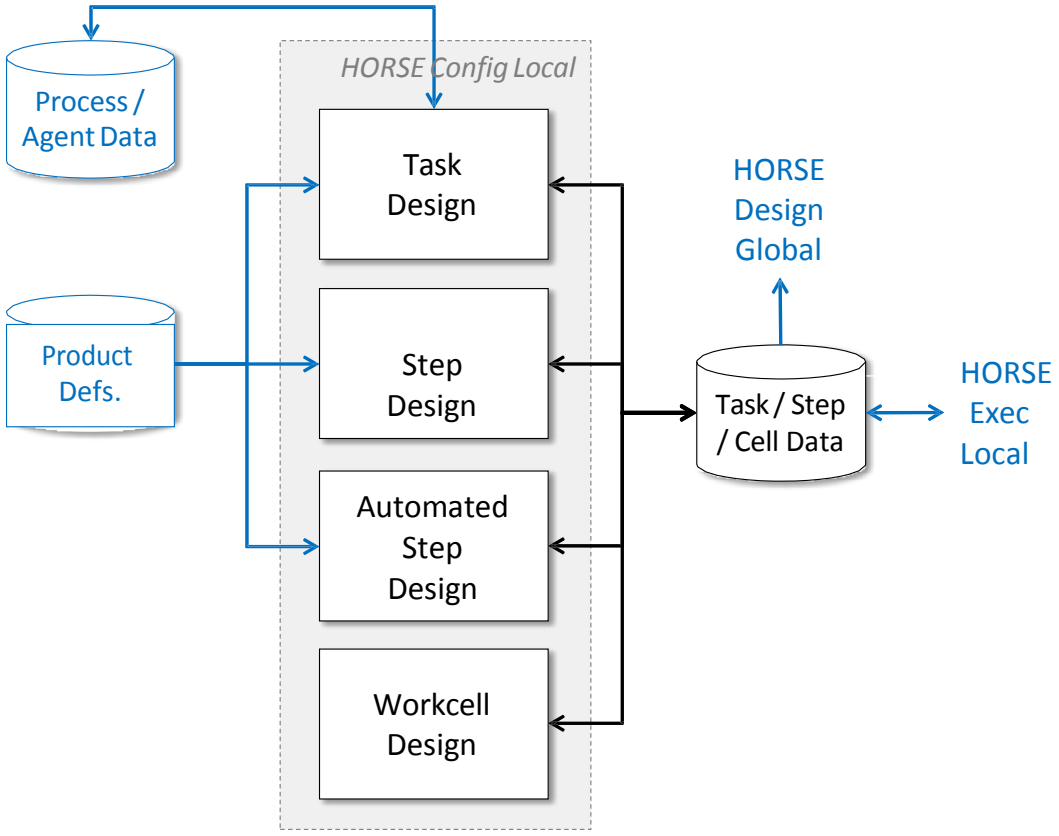

Figure 21: logical software architecture aggregation level 3, subsystem 3

6.4.4 HORSE Exec Local

The HORSE Exec Local subsystem is responsible for manufacturing activities within individual work cells, i.e., at the work cell level (see Figure 14). This involves two main functions:

- supporting execution of manufacturing tasks and steps, i.e., making things happen;
- supporting awareness about the state of execution, i.e., observe things that happen and processing this into relevant signals for controlling execution.

Both main functions are allocated to logical software modules at this aggregation level. Note that this design decision is isomorphic to the one for the HORSE Exec Global subsystem (see Section 6.4.2).

It is important that design decisions w.r.t. the internals of the HORSE Exec Local module (either in the logical architecture or later in the development and process architectures) are as much as possible isolated from design decisions in the HORSE Exec Global module. For this reason, we include an abstraction layer (Exec Local Abstraction layer) in the interface to the HORSE Exec Global

subsystem. Note that this decision is symmetric w.r.t. the design of the HORSE Exec Global subsystem (see Section 6.4.2).

The above decisions lead to the logical HORSE Exec Local architecture shown in Figure 22. The architecture also shows the contextual connections following from the software architecture at Level 2 (see Figure 15).

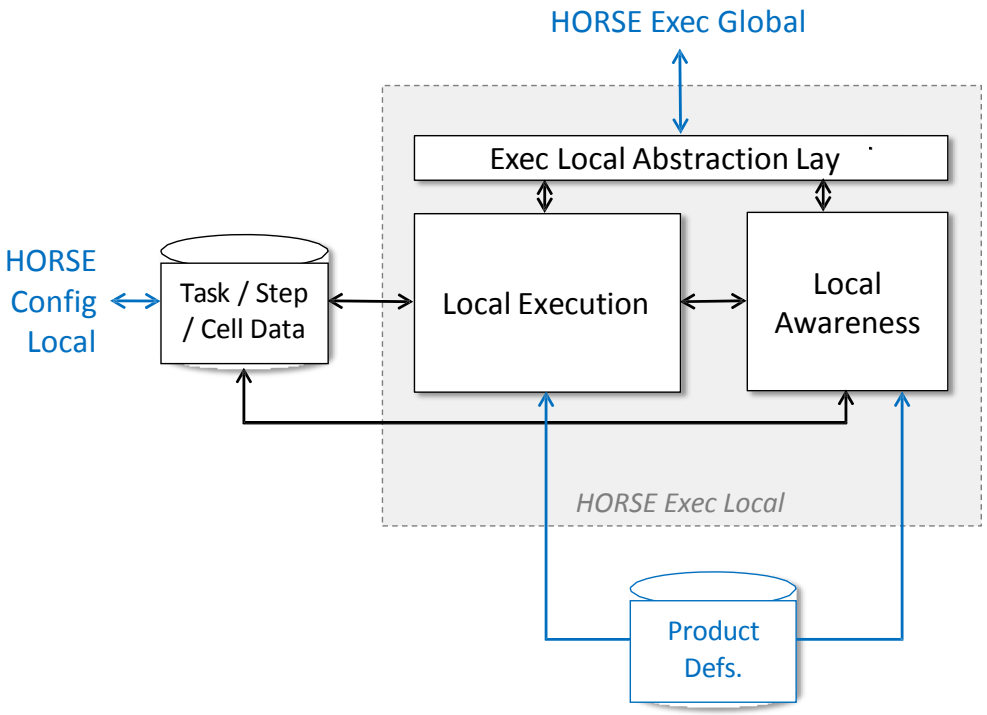

Figure 22: logical software architecture aggregation level 3, subsystem 4

6.4.5 Integration of subsystems

In Figure 23, we show the integration of the architectures of the four subsystems as developed in the previous four subsections, which is the overall logical software architecture at aggregation level 3. To not overcomplicate the figure, we have omitted the Product Definitions database and connections to it. For the same reason, we have also omitted interfaces to the hardware platform (robots, sensors) and human operators - these interfaces are shown when we refine this architecture to aggregation level 4 (in Sections 11 to 14 in this document).

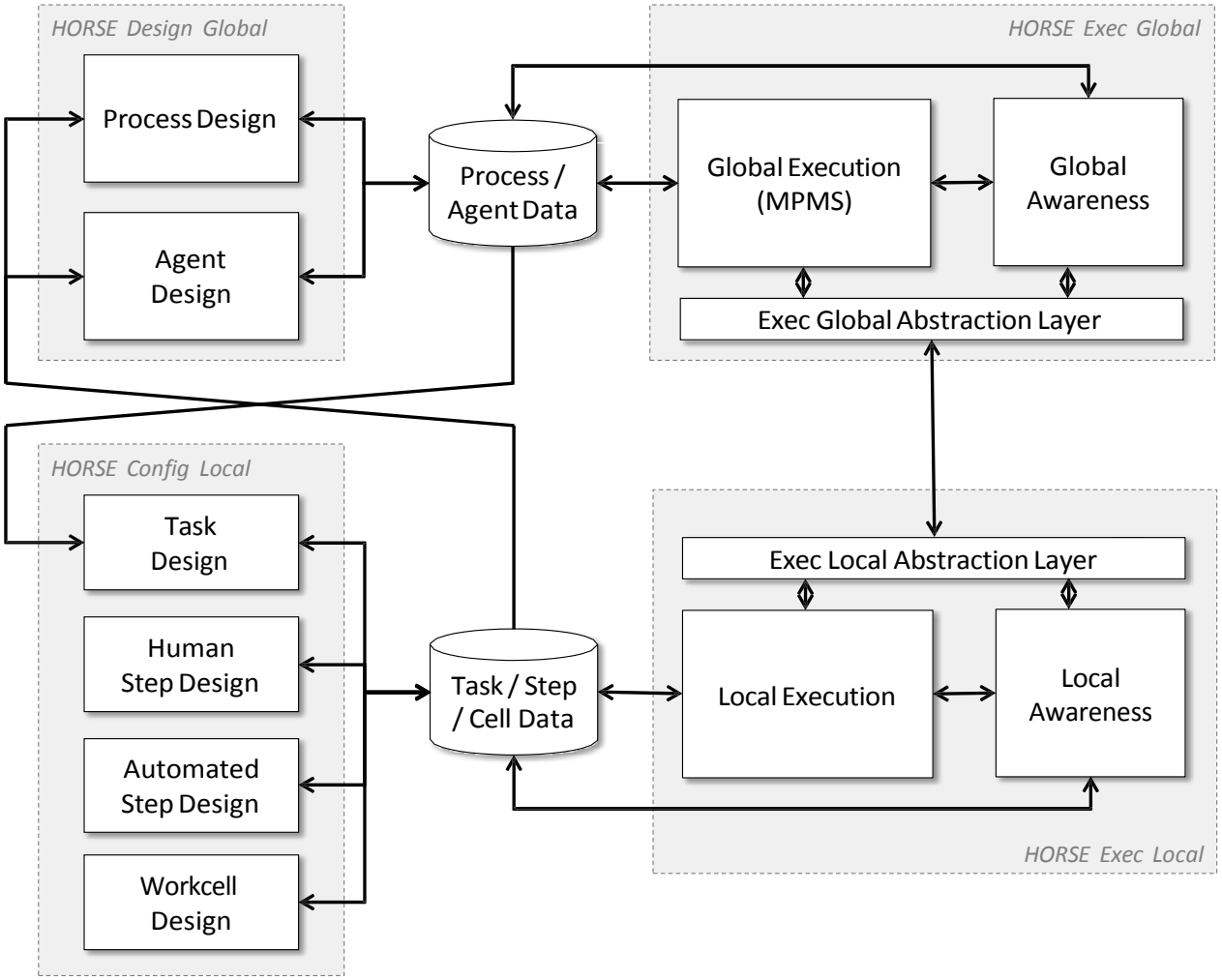

Figure 23: overall logical software architecture, aggregation level 3 (Product Defs. DB and interfaces to platform/operators omitted)

6.4.6 Confrontation with requirements

To check the architecture of Figure 23, we confront this logical software architecture with the HORSE system requirements [HOR16]. Given the level of detail of the architecture, we confront it with the top-down functional requirements at level 3. We check again for completeness and minimality, as explained before. The result of the confrontation is shown in Table 3.

		HORSE Design Global		HORSE Exec Global		HORSE Config Local				HORSE Exec Local	
		Process design	Agent design	Global execution	Global awareness	Task design	Human step design	Automated step design	Workcell design	Local execution	Local awareness
AF-01: Situation awareness	SF01: Work cell situation awareness				X					X	X
	SF02: Visual detection of item, intrusion, obstacle										X
AF-02: Synchronization of robotics and human activities	SF03: Grip, lift, handle, manipulate and release items (…)								X	X	
	SF04: Task allocation between human and automated actors (…)	X		X	X	X	X	X			
	SF05: Ergonomic work station supporting user friendly interaction					X	X		X		
	SF06: Force control enabling low inertia systems									X	
	SF07: Monitoring and intention estimation of human actions								X	X	X
AF-03: Robotics task instr	SF08: Robot programming by demonstration							X		X	
AF-04: Human task instructions	SF09: Augmented reality for manufacturing tasks						X		X	X	X
	SF10: Augmented reality to assist visual inspection						X		X	X	X
PF-01: Horizontal BPM	SF11: Manufacturing process management and monitoring	X		X							
	SF12: Structured manufacturing process exception handling	X		X							
PF-02: Resource mgmt.	SF13: Task allocation based on resource characteristics		X	X							

PF-03: Actor Control	SF14: Monitoring and control of work progress (…)			X							

Table 3: confrontation with top-down requirements Level 3

Table 3 shows that:

a) All Level 3 system requirements [HOR16] are covered by the logical software architecture at aggregation Level 3; thus the architecture is complete at this level.

b) All modules in the logical software architecture at aggregation Level 3 have a functionality linked to a Level 3 requirement; thus the architecture contains no superfluous modules.

Consequently, we observe that the confrontation of the architecture at Level 3 with the requirements at level 3 is successful.

7 HORSE logical data architecture

Before we can continue the further structural decomposition of the logical architecture, we must pay attention to the logical data architecture used in HORSE. The reason for this is that the concepts defined in the logical data architecture determine part of the logical software structure at the detailed level.

To specify logical data architectures, we use UML class diagrams [Wik16a].

Below, we develop a set of HORSE concept models:

- an *agent concept model*, which specifies the concepts and relations between concepts that describe actors in a manufacturing context, i.e., entities that can perform manufacturing activities;
- an *activity concept model*, which specifies the concepts and relations between concepts that describe the activities to be performed in a manufacturing context by agents;
- an *event concept model*, which specifies the concepts and relations between concepts that describe events that require reactions in a manufacturing context; this concept model is included because monitoring and safety are important aspects in HORSE.

We describe each concept model in the three following subsections. In Section 7.4, we show how the concept models can be integrated into the *HORSE overall concept model*.

The terminology used in the concept models is HORSE-specific. Linking to broadly accepted terminologies (like those of IEC [IEC13], Industrie 4.0 [GTI14] and OMG) is desired, however.

7.1 HORSE agent concept model

The HORSE agent concept model is shown in Figure 24 and explained in the following. Note the UML class diagram notation:

- Lines without heads denote general relationships (with indicated cardinalities).
- Lines with a diamond head denote part-of relationships (with indicated cardinalities, the part at the diamond head side).
- Lines with an arrow head denote subtyping relationships (the supertype is at the arrow head side).

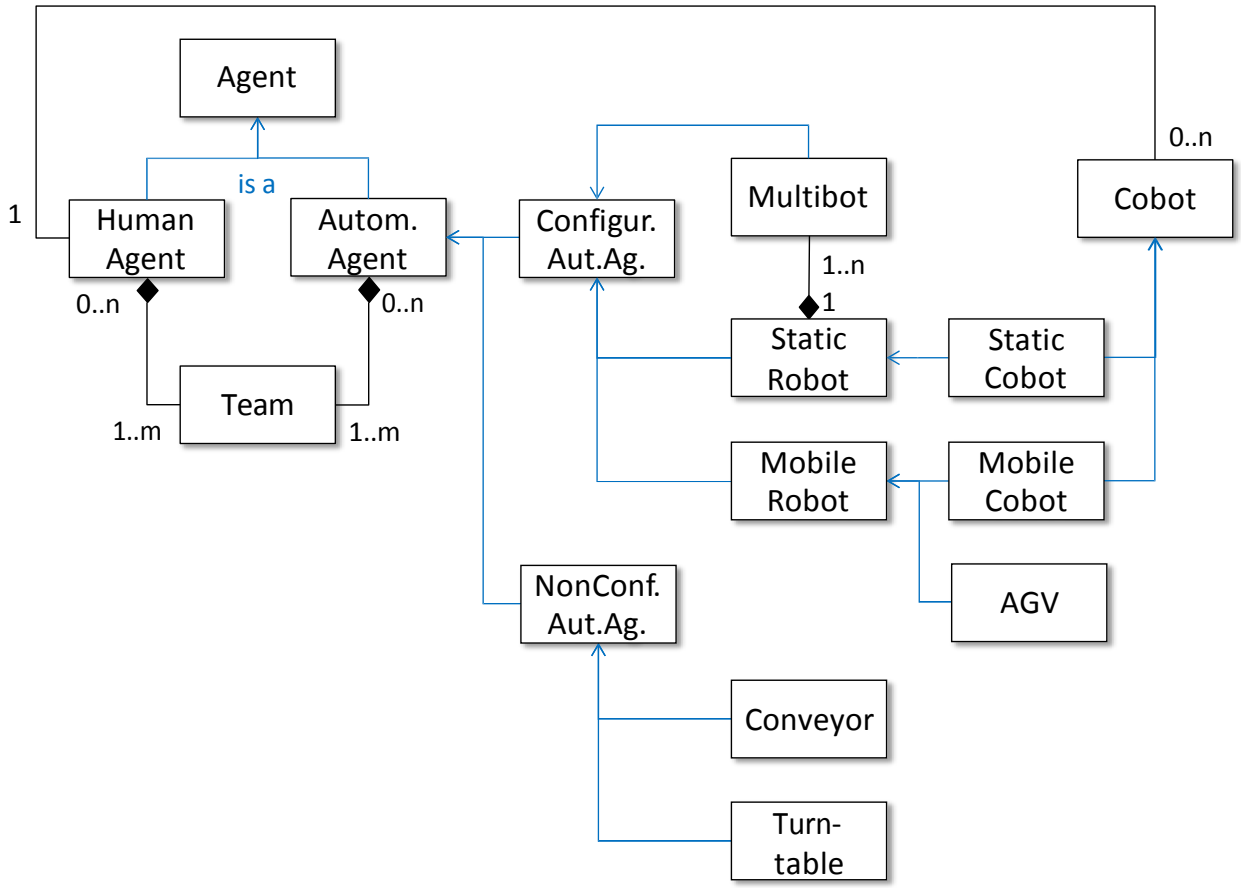

Figure 24: HORSE agent concept model

The central concept is *Agent*, which represents anything that can perform a step in manufacturing. An *Agent* can be part of one or more *Teams*.

The *Agent* concept is subtyped into *Human Agent* and *Automated Agent*. The *Automated Agent* is further subtyped into *Configurable Automated Agent* (programmable robot) and *NonConfigurable Automated Agent* (other non-programmable active manufacturing equipment). The Configurable *Automated Agent* concept is again subtyped into *Static Robot* (fixed at one position in a work cell) and *Mobile Robot* (able to move around in a manufacturing location).

Several *Static Robots* can be combined into one *Multibot*. A *Multibot* is programmed as one entity, i.e., a *Multibot* is a *Configurable Automated Agent* and not a *Team*.

A *Cobot* is a specific kind of *Robot* and can be either a *Static Cobot* or a *Mobile Cobot*. A *Cobot* is associated to a *Human Agent*.

The HORSE agent concept model includes the following constraint:

- A team includes at least one agent (empty teams are not permitted).

7.2 HORSE activity concept model

The HORSE activity concept model is shown in Figure 25 and explained in the following.

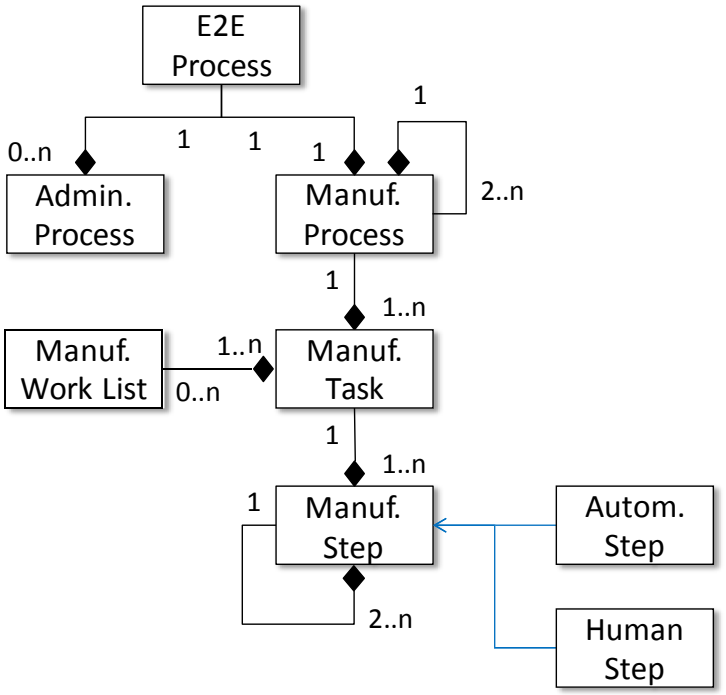

Figure 25: HORSE activity concept model

A *Manufacturing Process* can exist of two or more sub processes.

Within a *Manufacturing Process* we have *Manufacturing Tasks* that coincide with work cells.

Within a *Manufacturing Task*, Manufacturing *Steps* are performed. *Manufacturing Steps* may have sub steps as well. Because we have human and automated agents we also have *Human Steps* and *Automated Steps*.

The *Manufacturing Process* is part of an *End-to-End Process*. An *End-to-End Process* consists of zero or more *Administrative Processes* and one *Manufacturing Process*. The concept of *End-to-End Process* is outside the strict scope of the HORSE system, but scope-wise coincides with the system context architecture as shown in Figure 11.

7.3 HORSE event concept model

The HORSE event concept model is shown in Figure 26 and explained in the following. Note that the event concept model contains only one general relationship (between *Use* and *Event*), all other links represent subtyping relations.

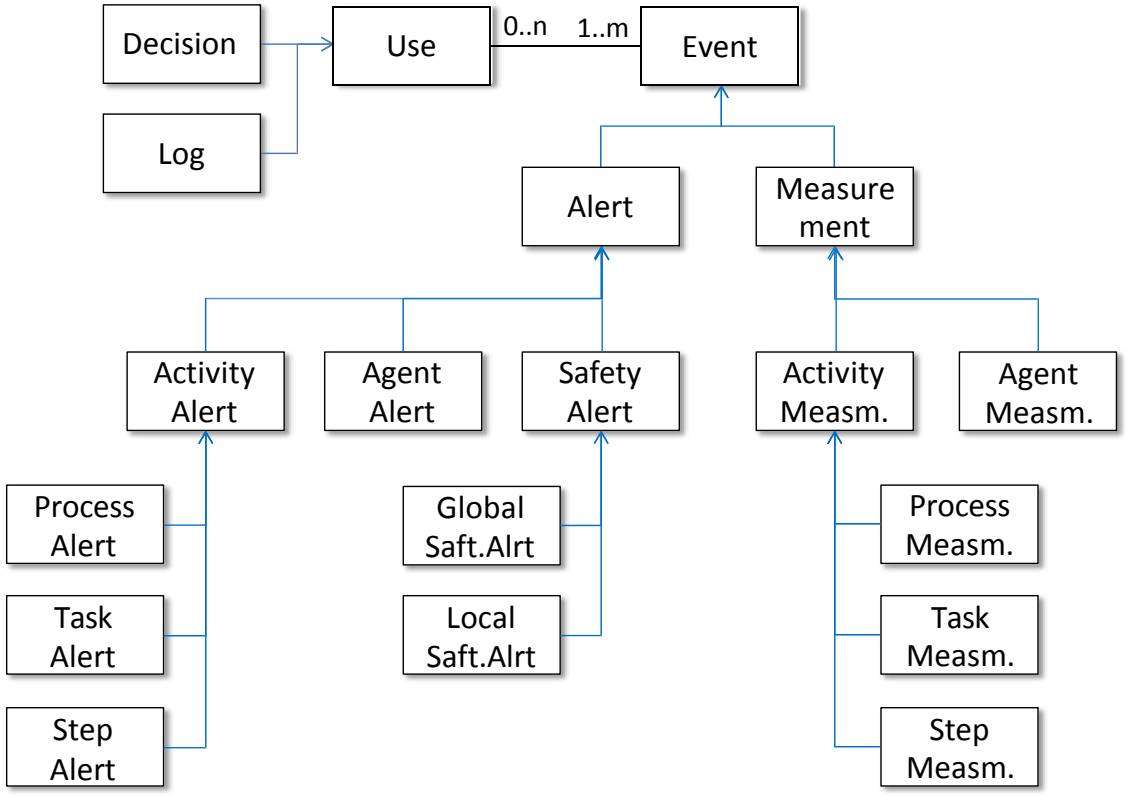

Figure 26: HORSE alert concept model

The HORSE event concept model is built around the central concept of *event*. The concept *event* is specialized into the concepts of *alert* and *measurement*. An *alert* is an event generated in an exceptional, unplanned situation. A *measurement* is a planned, periodic event that generates data. An *event* is coupled to its *use*. In the *use*, it is processed, e.g. to make a *decision* or to store data in a *log*.

The concept *alert* is specialized into three subtypes:

- *activity alert*: an alert generated by a system module that executes an activity at the level of process, task or step as defined in the HORSE activity concept model, irrespective of the agent(s) involved in the activity; an example is a manufacturing step exceeding its maximum execution time;
- *agent alert*: an alert generated by an agent as defined in the HORSE agent model, irrespective of the activity the agent is performing at that moment; an example is a mobile robot predicting a collision;
- *safety alert*: an alert generated by an observed safety breach at the global level (site, area or production line as defined by the IEC standard of Figure 5) or local level (work cell level), irrespective of the activities or agents involved; an example is the manufacturing hall temperature exceeding a threshold value.

7.4 Overall concept model

The connection between the HORSE agent concept model (Figure 24), the HORSE activity concept model (Figure 25) and the HORSE event concept model (Figure 26) is shown in Figure 27. This figure shows part (a projection) of the HORSE overall concept model - details not relevant for the mentioned connection have been omitted for reasons of clarity.

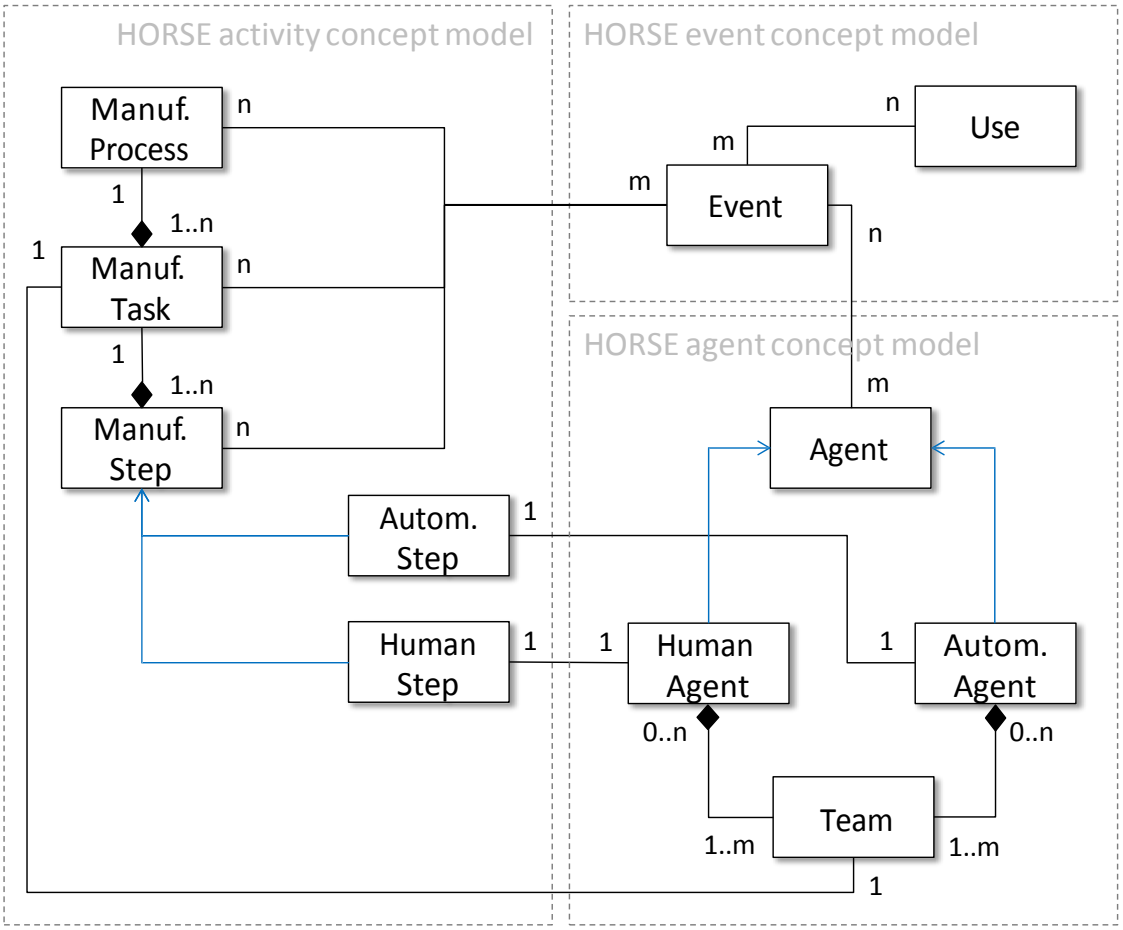

Figure 27: projection of overall concept model

8 HORSE logical organization architecture

The organization aspect of the HORSE logical architecture contains the structure of the organizational functions involved in the management, operation and use of a HORSE system. It is meant to be a descriptive architecture, not a prescriptive architecture (in concrete cases, the organization may be different). It can be used as a sanity check to see whether a pilot scenario design corresponds to the manufacturing organisation's organogram.

8.1 Abstract logical organization architecture

Figure 28 shows the HORSE organization architecture. Grey elements denote departments, white elements denote roles. The roles coincide with those in the high-level HORSE scenario elaborated in Section 5.

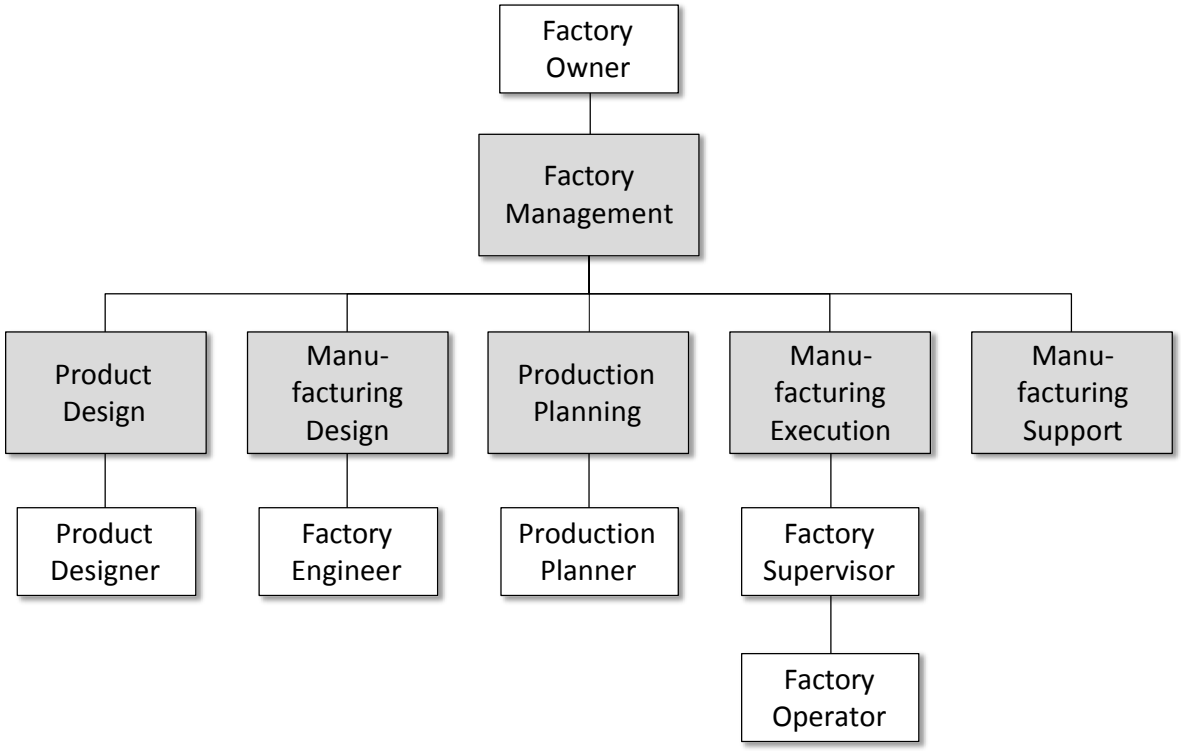

Figure 28: HORSE organization architecture

8.2 Mapping to IEC standard

To make a clear mapping of the organization architecture to the IEC standard hierarchy (as discussed in Section 4.2), we have to bring the levels of the IEC standard into the Manufacturing Execution part of the organization architecture. We can do this by specializing the *Factory Supervisor* role as shown in Figure 29 into three levels: *site supervisor*, *area supervisor*, and *line supervisor*. For reasons of brevity, we do not include these roles in the abstract scenarios as specified in Section 5. They can be used in concrete, case-specific scenarios, however, if so required.

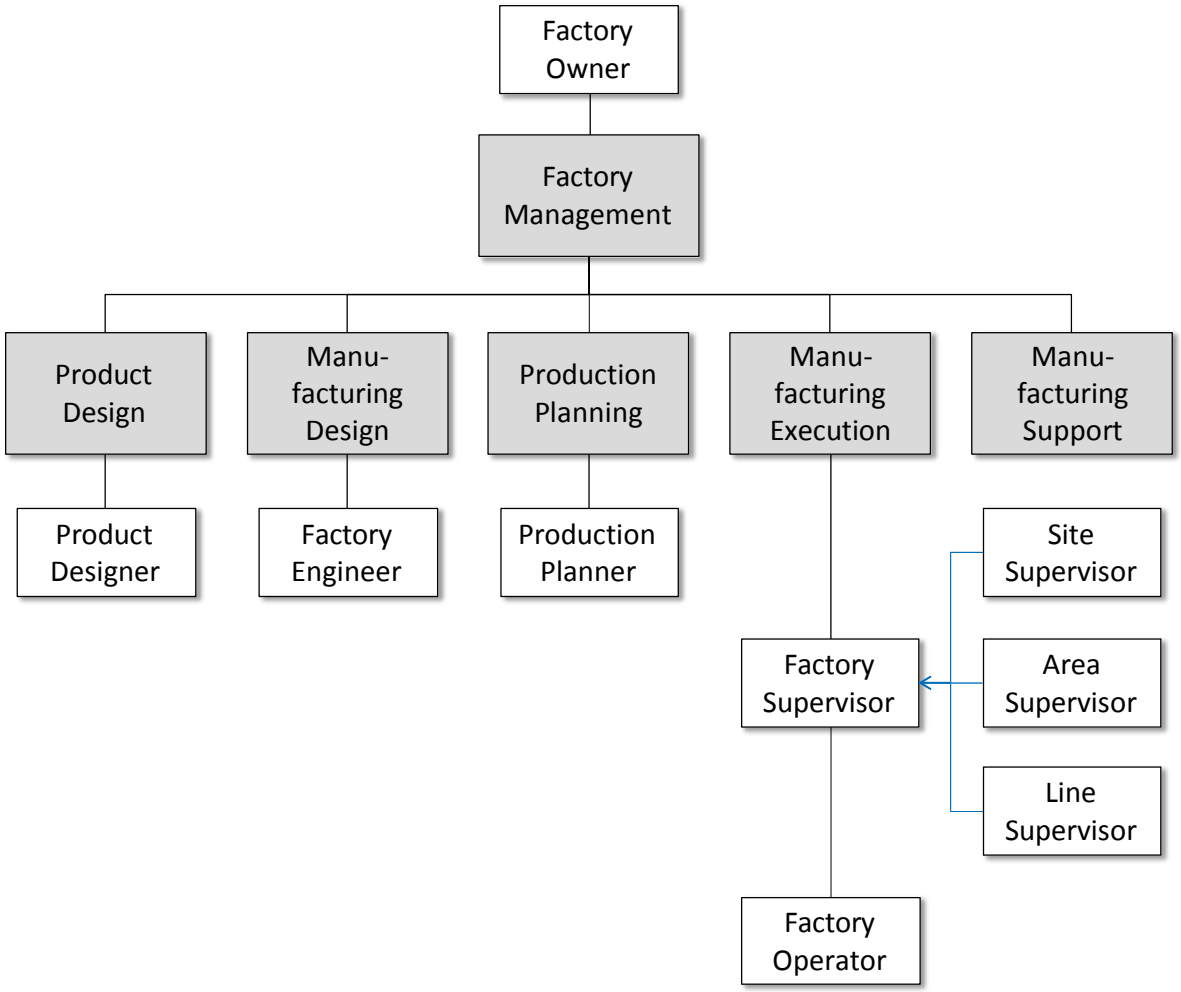

Figure 29: HORSE organization architecture extended to match IEC hierarchy

With this extended organization architecture, we can make the mapping as shown in Figure 30. To not overcomplicate the figure, we project on the relevant part of the architecture (leaving out four of the five branches of Figure 29).

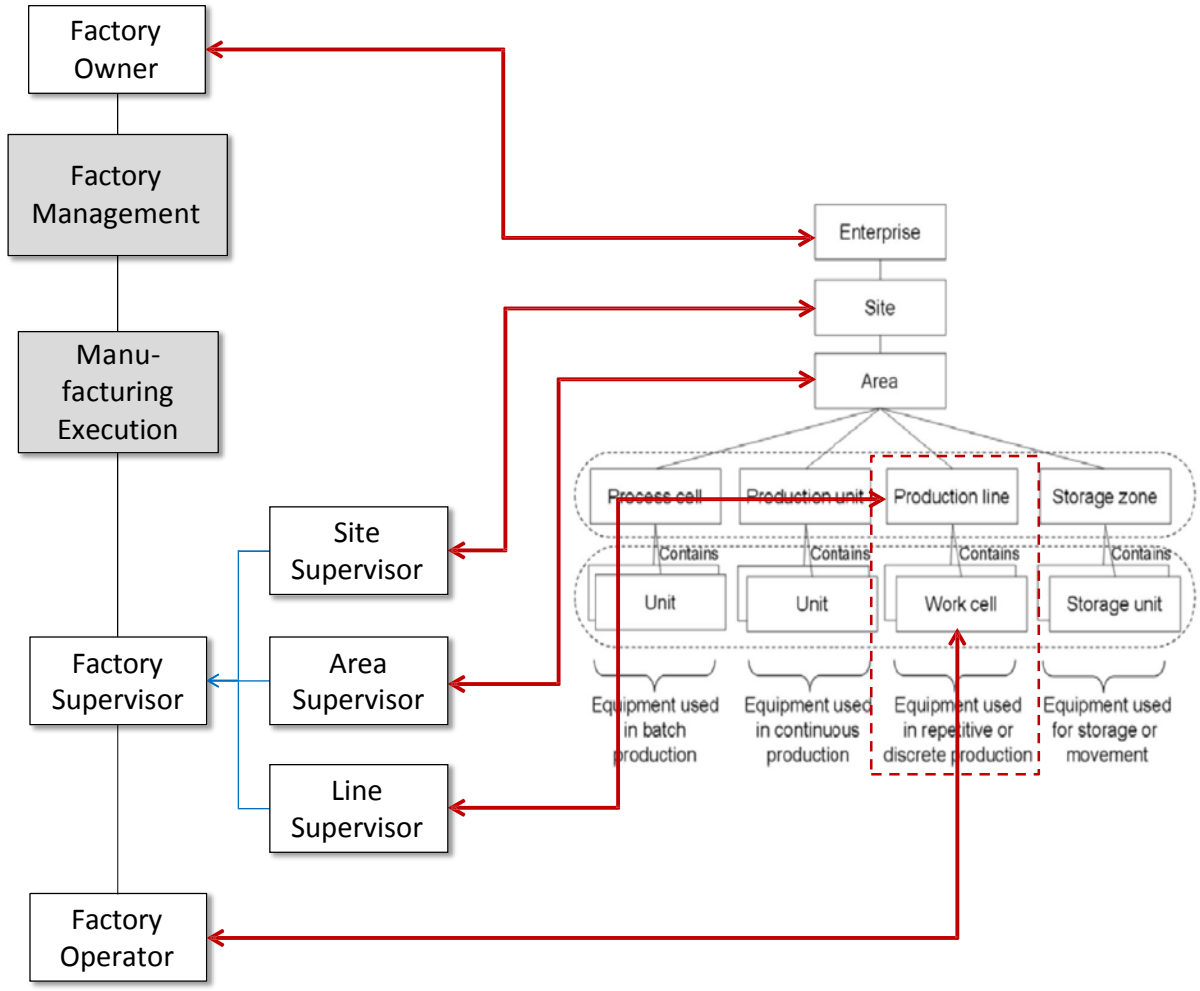

Figure 30: projection of extended HORSE organization architecture mapped to IEC hierarchy

The mapping of Figure 30 can be used to map the organization architectures of the three pilot cases via the IEC hierarchy to the logical HORSE organization architecture in Figure 29.

9 HORSE logical process architecture

This section describes the process aspect of the HORSE logical architecture, i.e., the structure of manufacturing processes supported by the HORSE system.

The conceptual basis for the process architecture is established in the data architecture, as this architecture contains the HORSE activity concept model (see Section 7.2). In this section, we make this activity concept more concrete. We do this at two levels: the level of enterprise processes to make the positioning of manufacturing processes clear, and at the level of manufacturing processes to make the internal structure of these processes clear.

9.1 Enterprise process level

At the enterprise process level, we show two cases: one for custom-designed production and one for series production.

9.1.1 Custom-designed production

In Figure 31, we see a simplified example end-to-end enterprise process for delivering a custom-designed product.

At level 1, we have the top level that shows the sequencing of the main enterprise activities (which may be related to a value chain model like Porter's [Port85]). Note that we have incorporated two options: (1) a product specification is designed from scratch; (2) a product specification is retrieved from a catalog (and possibly modified).

At level 2, we see the subprocesses, which are refinements of the steps at level 1. The sub-process to the left is an administrative process, the one to the right a manufacturing process (as part of the conceptual activity model of Figure 25). The manufacturing process is the core scope of HORSE (as discussed in Section 6.1).

The manufacturing process contains a step to (re)design the manufacturing process, a step to configure one or more work cells, and two actual manufacturing steps.

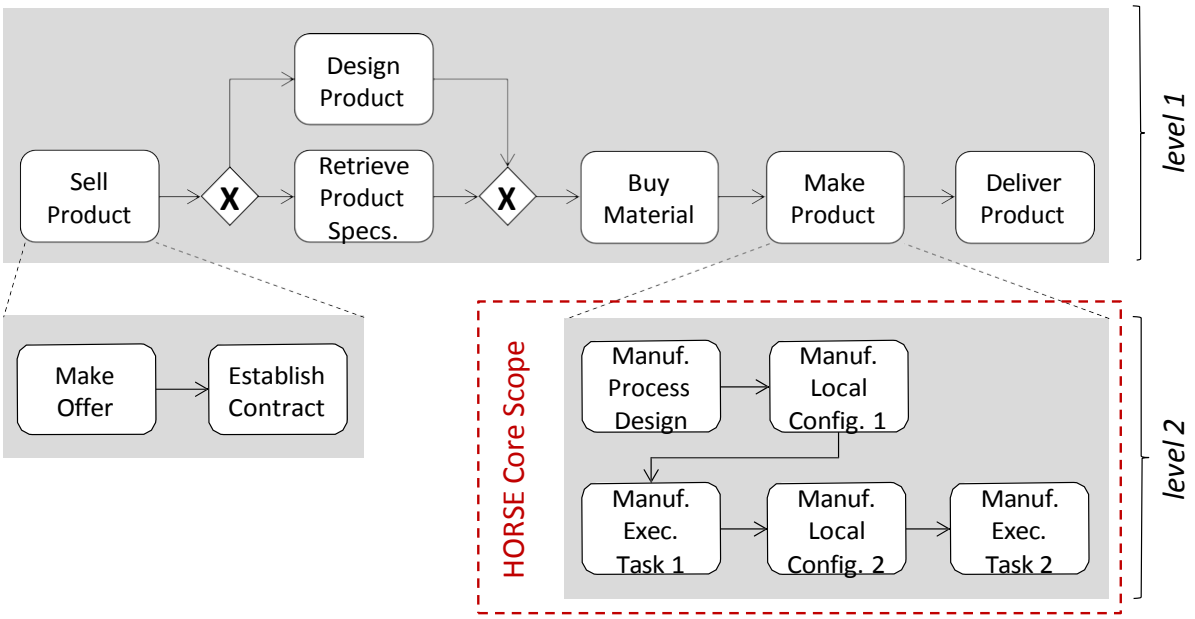

Figure 31: example enterprise process for custom-designed production

9.1.2 Series production

In Figure 32, we see a variation of the enterprise process for series products. In the beginning of the process, a design step is included at Level 1. In its sub-process at Level 2, we see a product design step (not in the core scope of HORSE) and a manufacturing process design step (in the core scope of HORSE). The other steps at Level 1 are included in an iteration, which represents the handling of a batch of products. The *Make Product* step has again a sub-process at Level 2. This sub-process includes a configuration step for one or more work cells, such that batches of various product types can be interleaved in production.

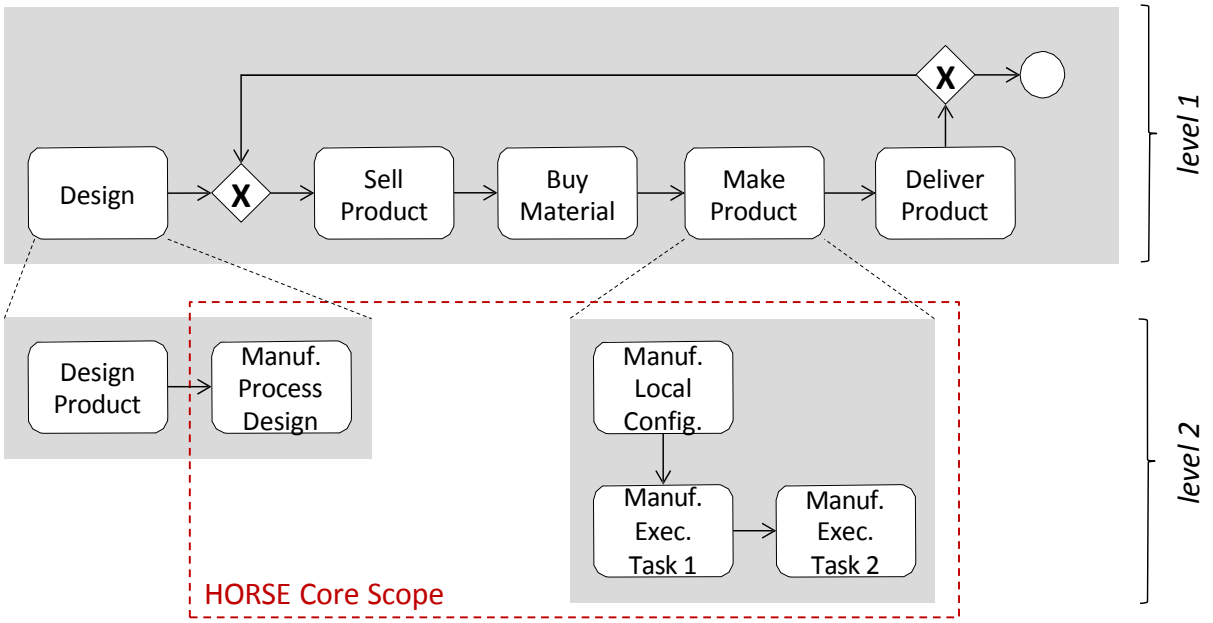

Figure 32: example enterprise process for series production

9.2 Manufacturing process level

In Figure 33, we see a refinement of the manufacturing process of Figure 31. Conforming to the activity concept model of Figure 25, the tasks consist of steps, which can consist of sub-steps again. Note that at the highest level, design time and execution time steps can be interleaved.

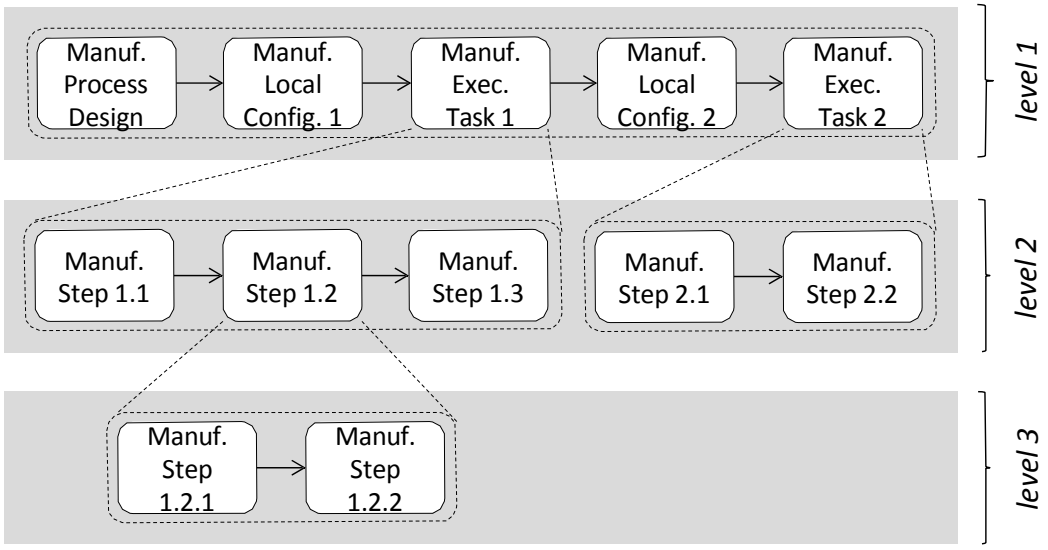

Figure 33: example manufacturing process with tasks and hierarchic steps

Process models like the one in Figure 33 have been practically elaborated for the three HORSE pilot cases in the HORSE requirements analysis [HOR16].

10 HORSE logical platform architecture

This section describes the platform aspect of the HORSE logical architecture, i.e., the structure of the software and hardware systems that form the basis for the operation of the HORSE system.

10.1 Software platform

The software platform includes:

- Standard business process management software.
- Database management software.
- Middleware to connect the above systems and HORSE modules.
- Middleware to connect HORSE modules to hardware interface software.
- Hardware interface software.

10.2 Hardware platform

The hardware platform includes:

- Robots.
- Other automated agents, such as conveyors.
- Sensors, including cameras.
- Computers.

10.3 Platform overview

The platform architecture is shown in overview in Figure 34.

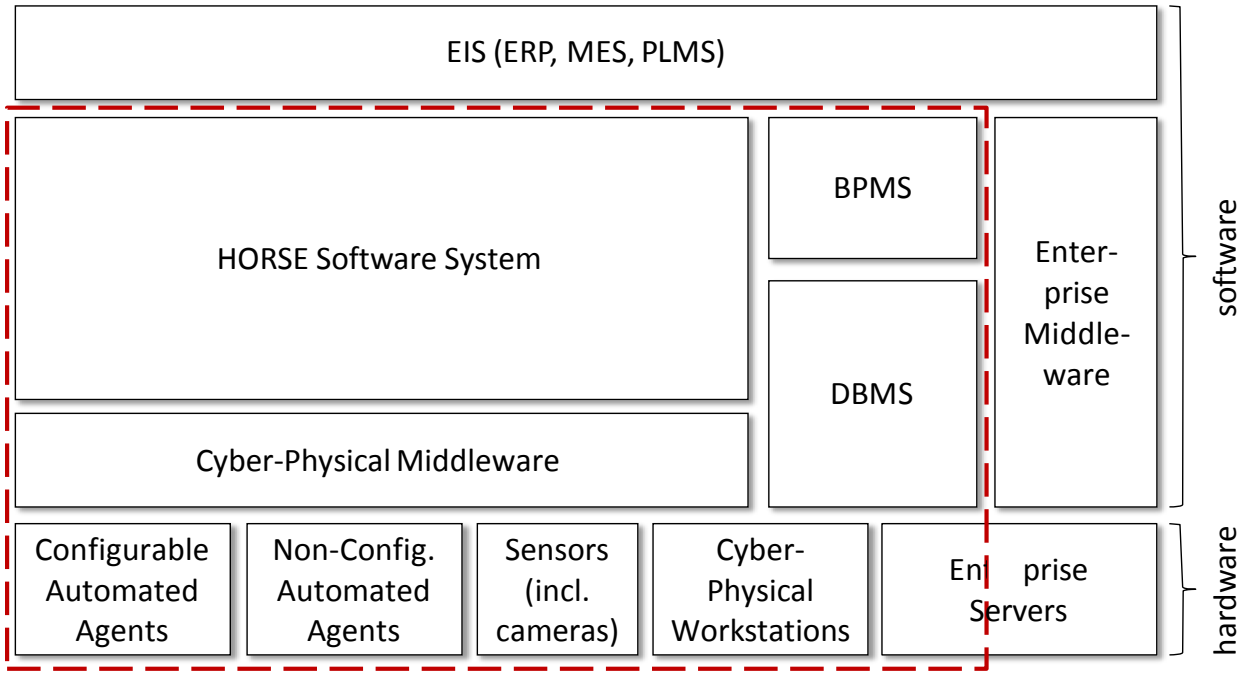

Figure 34: HORSE platform architecture (with HORSE focus indicated)

HORSE
Complete System Design
Part 3:

Medium-Level Design of the Software Aspect

11 Logical software architecture HORSE Design Global

This section describes the logical software architecture at aggregation level 4 of the HORSE Design Global module, i.e., it refines the logical software architecture design of Section 6.4.1 (as shown in Figure 19). First, we elaborate the individual sub-modules of the HORSE Design Global module. Then we integrate these elaborations to get an overview of the HORSE Design Global module at aggregation level 4. A hierarchical list of all modules in the logical software architecture with hierarchical module IDs is included in Appendix C of this document.

11.1 Process Design module

The Process Design module contains the functionality to (re-)design manufacturing processes. Results of design activities are stored in the Process/Agent Definitions database. In case of redesign, the input is retrieved from this database.

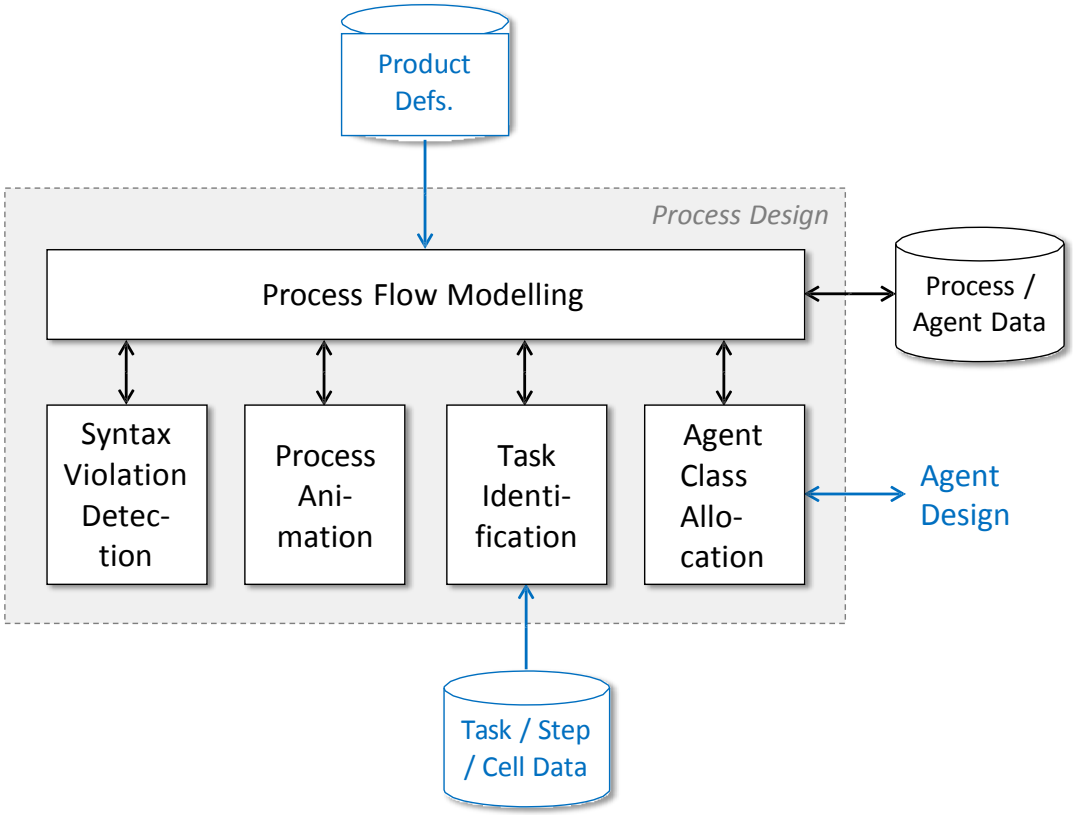

Figure 35: Process Design module software architecture, aggregation level 4

As shown in Figure 35, the module consists of five sub-modules.

11.2 Agent Design module

The Agent Design module contains the functionality to design manufacturing agents, i.e., describe their relevant characteristics.

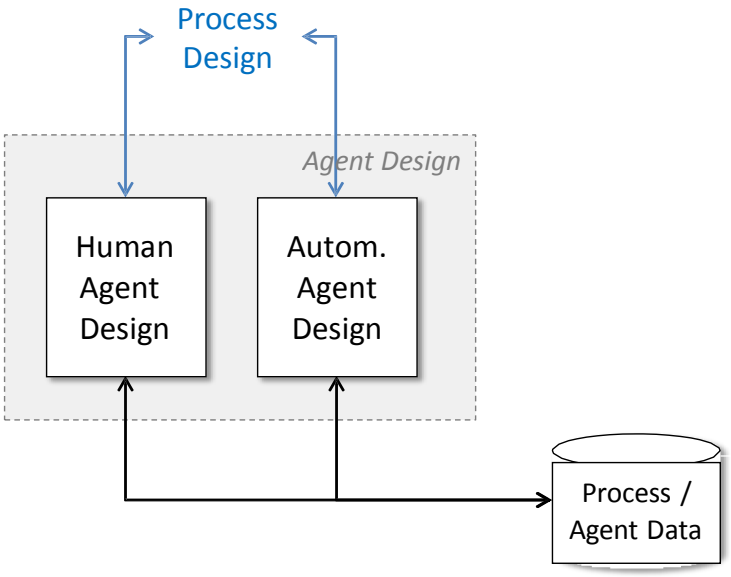

Figure 36: Agent Design module software architecture, aggregation level 4

11.3 HORSE Design Global overview (aggregation level 4)

In Figure 37, we see the integration of the architectures of Figure 35 and Figure 36, i.e., the overview of the HORSE Design Global software architecture at aggregation level 4.

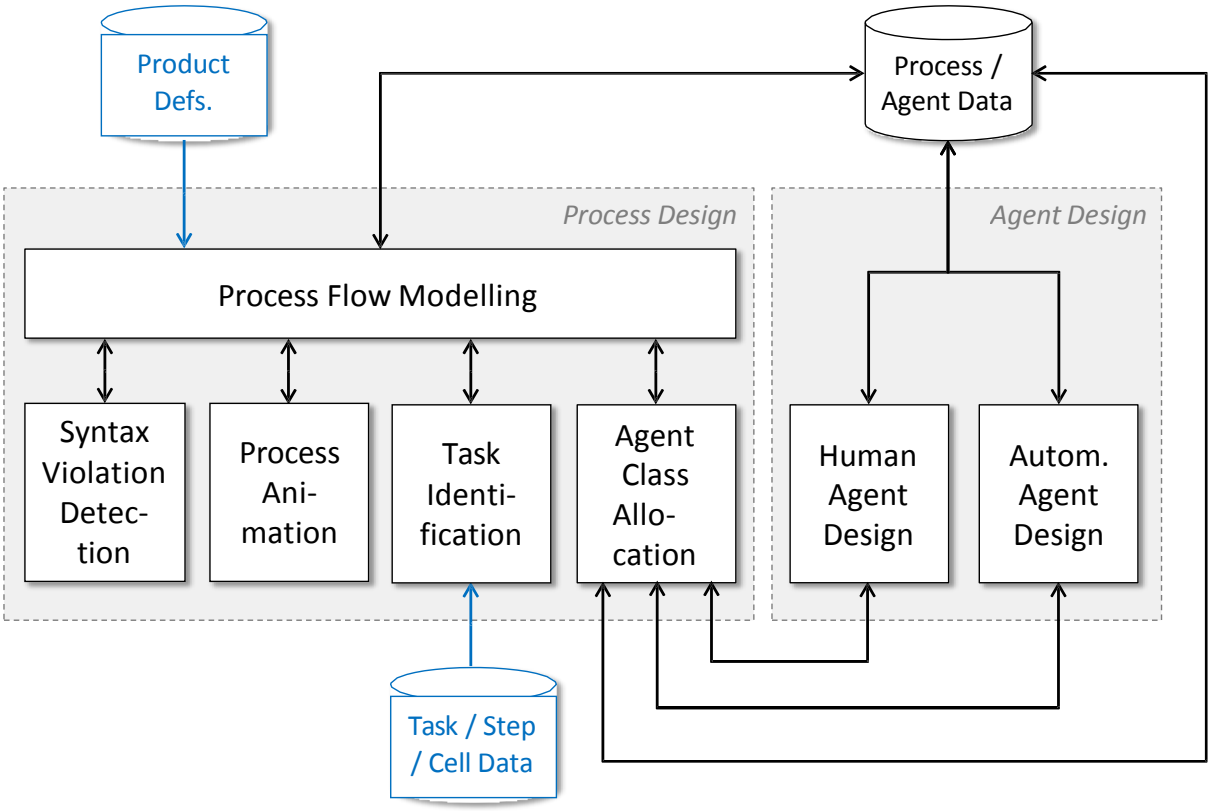

Figure 37: HORSE Design Global logical architecture, aggregation level 4

12 Logical software architecture HORSE Exec Global

This section describes the logical software architecture at aggregation level 4 of the HORSE Exec Global module, i.e., it refines the logical software architecture design of Section 6.4.2 (as shown in Figure 20). First, we elaborate the individual sub-modules of the HORSE Exec Global module. Then we integrate these elaborations to get an overview of the HORSE Exec Global module at aggregation level 4. A hierarchical list of all modules in the logical software architecture with hierarchical module IDs is included in Appendix C of this document.

12.1 Global Execution module

The software architecture at aggregation level 4 of the Global Execution module is shown in Figure 38.

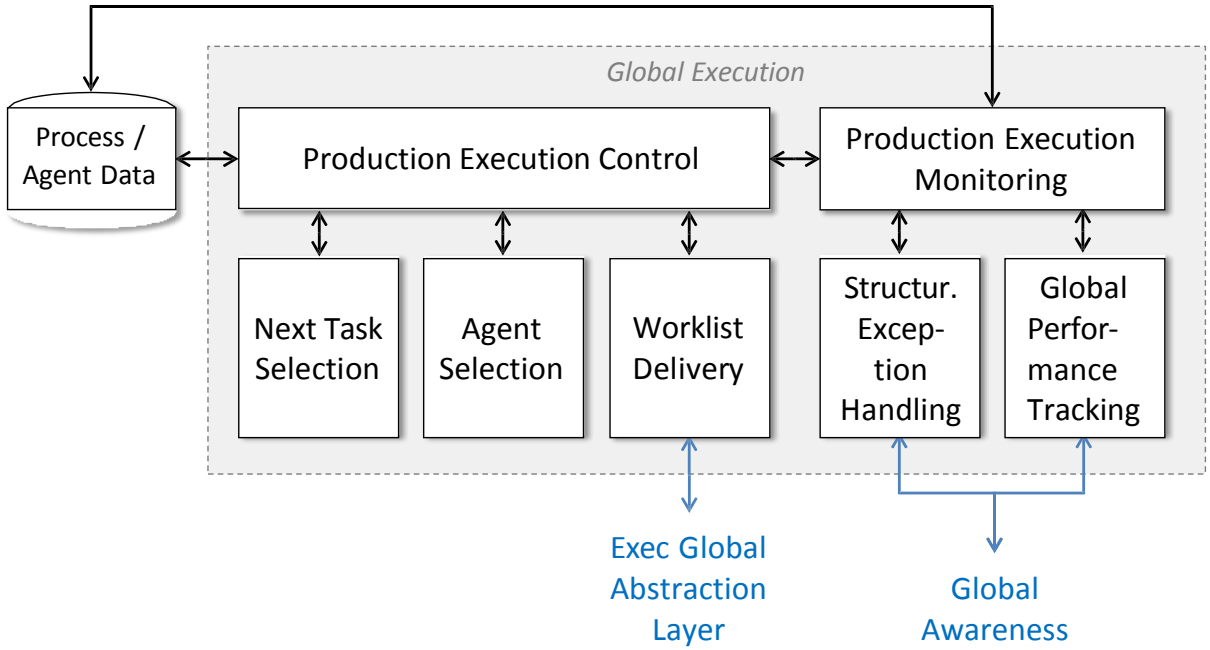

Figure 38: Global Execution module software architecture, aggregation level 4

The Worklist Delivery module supports push mode task delivery. Work Lists consist of a single task in the current system design for reasons of simplicity (but may contain multiple tasks in a future design - as shown in the HORSE activity concept model of Figure 25).

Production Execution Monitoring module supports real-time monitoring of manufacturing execution in terms of processes, orders, and agents (human and automated).

12.2 Global Awareness module

The aggregation level 4 software architecture of the Global Awareness module is shown in Figure 39.

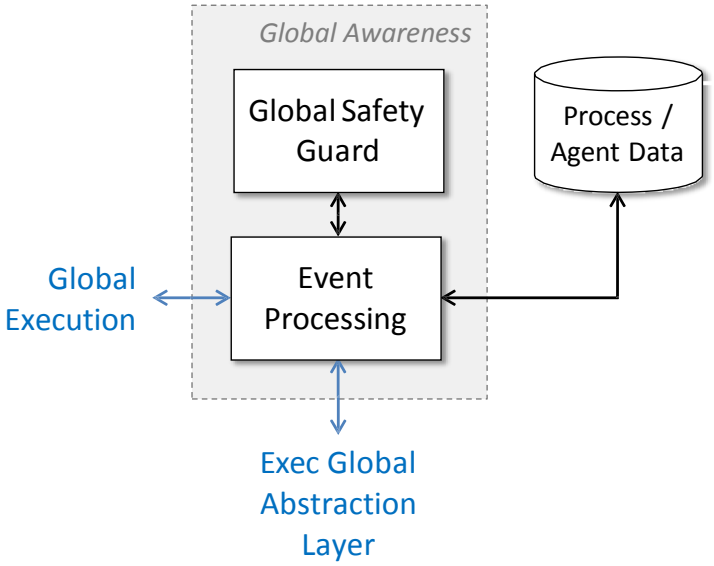

Figure 39: Global Awareness module software architecture, aggregation level 4

12.3 HORSE Exec Global overview (aggregation level 4)

Figure 40 shows the logical software architecture of the HORSE Exec Global subsystem at aggregation level 4. It is the integration of the architectures of Figure 38 and Figure 39.

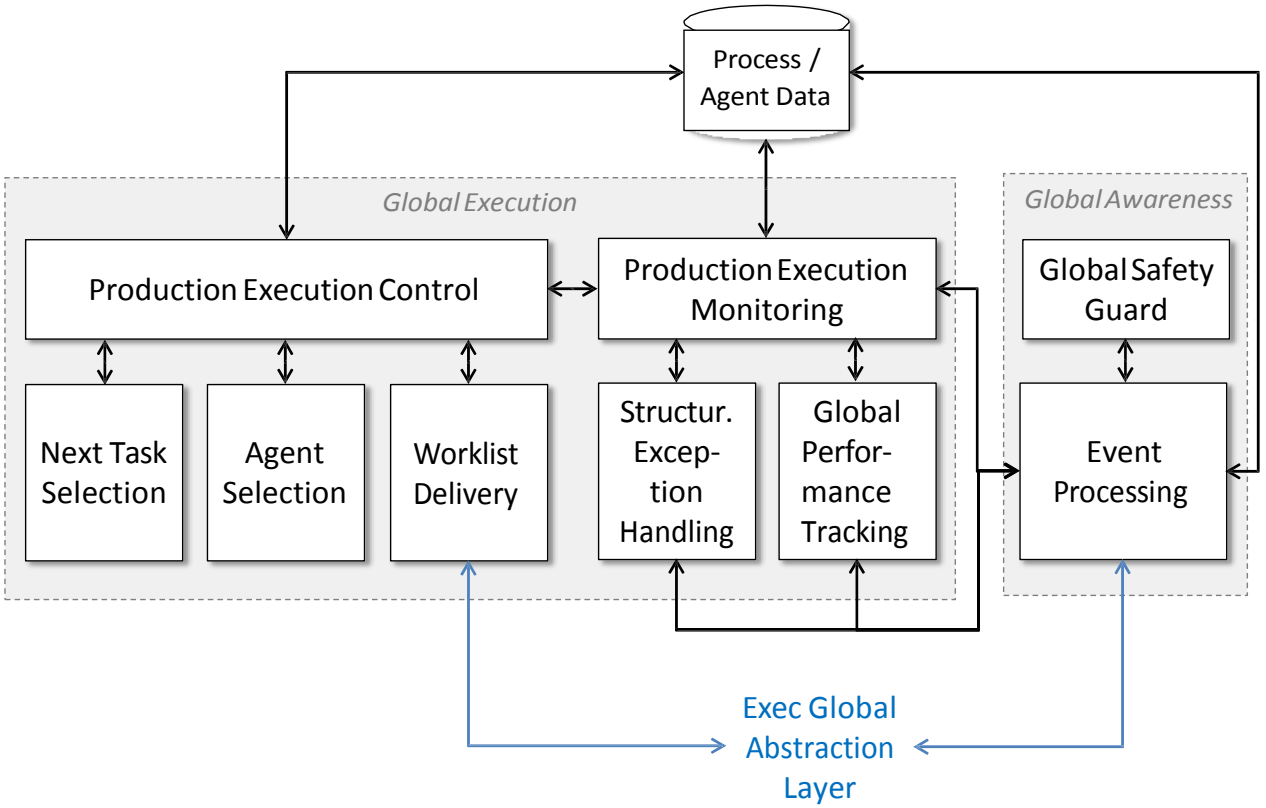

Figure 40: HORSE Exec Global logical architecture, aggregation level 4

13 Logical software architecture HORSE Config Local

This section describes the logical software architecture at aggregation level 4 of the HORSE Config Local module, i.e., it refines the logical software architecture design of Section 6.4.3 (as shown in Figure 21). First, we elaborate the individual sub-modules of the HORSE Config Local module. Then we integrate these elaborations to get an overview of the HORSE Exec Global module at aggregation level 4. A hierarchical list of all modules in the logical software architecture with hierarchical module IDs is included in Appendix C of this document.

Note that design GUIs are not included in the software architecture, as they are considered part of the platform architecture.

13.1 Task Design

Figure 41 shows the software architecture of the Task Design module.

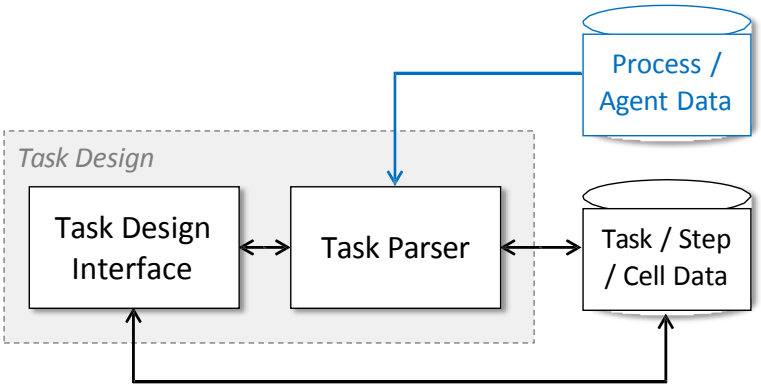

Figure 41: Task Design module software architecture, aggregation level 4

13.2 Human Step Design

Figure 42 shows the software architecture of the Human Step Design module.

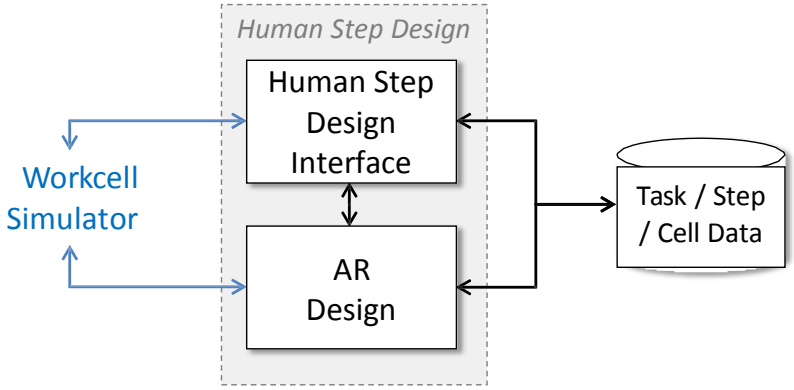

Figure 42: Human Step Design module software architecture, aggregation level 4

13.3 Automated Step Design

Figure 43 shows the software architecture of the Automated Step Design module.

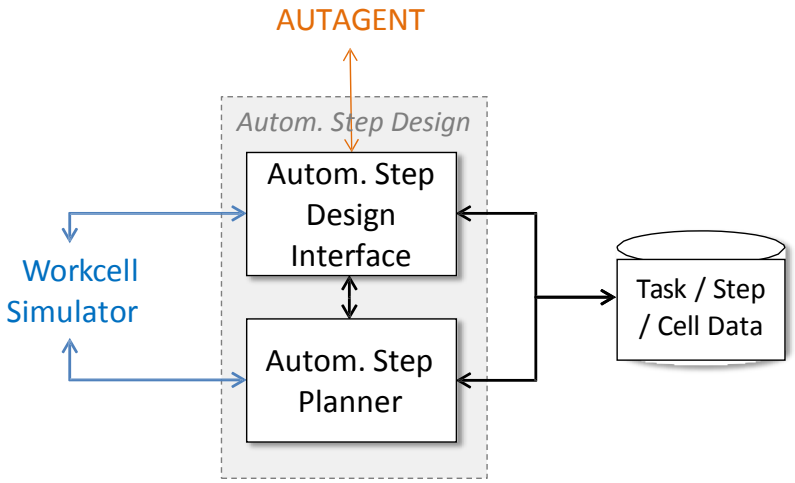

Figure 43: Automated Step Design module software architecture, aggregation level 4

As the module supports robot programming by demonstration, it contains an interface to an automated agent.

13.4 Workcell Design

Figure 44 shows the software architecture of the Workcell Design module.

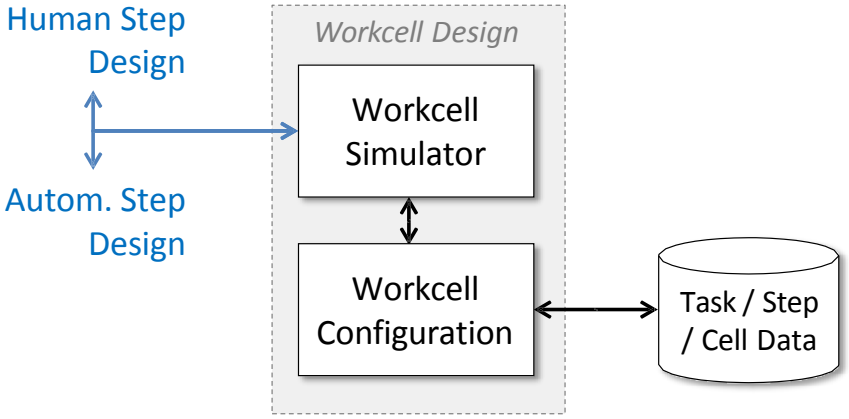

Figure 44: Workcell Design module software architecture, aggregation level 4

13.5 HORSE Config Local overview (aggregation level 4)

Figure 45 shows the logical software architecture at aggregation level 4 of the HORSE Config Local subsystem. This architecture is obtained by integrating the architectures of Figure 41, Figure 42, Figure 43 and Figure 44.

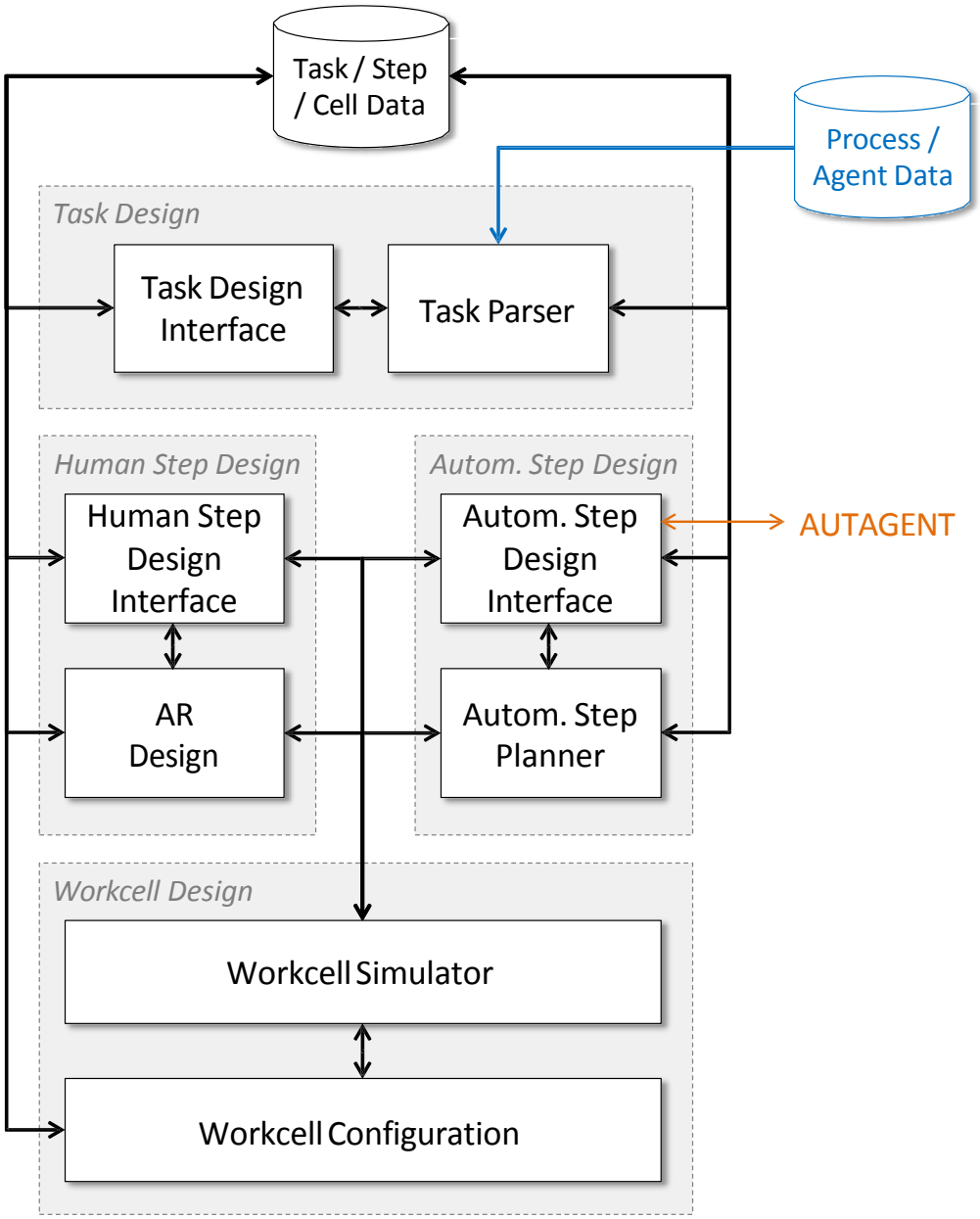

Figure 45: HORSE Config Local logical architecture, aggregation level 4

14 Logical software architecture HORSE Exec Local

This section describes the logical software architecture at aggregation levels 4 and 5 of the HORSE Exec Local module, i.e., it refines the logical software architecture design of Section 6.4.4 (as shown in Figure 22). A hierarchical list of all modules in the logical software architecture with hierarchical module IDs is included in Appendix C of this document.

Different from the other three main subsystems (as at aggregation level 2), the HORSE Exec Local subsystem is elaborated to aggregation level 5 because this subsystem contains the most detailed cyber-physical interfaces, which require specification in the logical architecture view to provide a solid basis for further elaboration in the development architecture view (in WP3).

14.1 Local Execution module

In refining the Local Execution module, we use the concept model described in Section 7.3 (as shown in Figure 27). In doing so, we link the software and data aspects of the logical architecture, as illustrated in Figure 46.

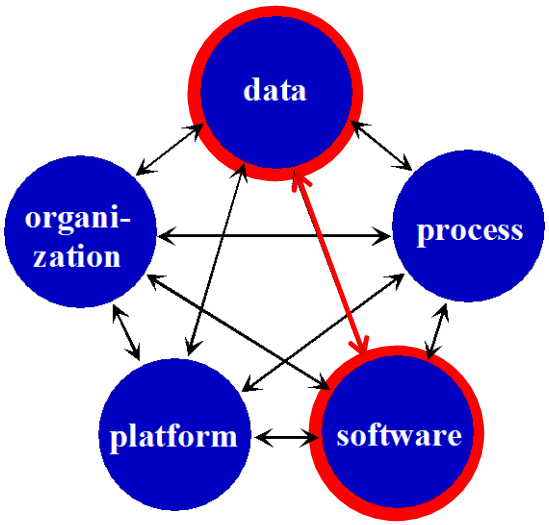

Figure 46: connecting software and data aspects

The decisions made there lead to decisions for the logical software architecture. We include a module (Hybrid Task Supervisor) that supervises the execution of a manufacturing task by a team of multiple agents (which can be of different types, such as humans, robots or different automated agents). We include separate modules for the supervision of execution of human respectively automated manufacturing steps: HumAgent Step Exec and AutAgent Step Exec.

Synchronization between these two modules is performed by the Hybrid Task Supervisor, as this module oversees the dependencies between agents in a task. These two modules have interfaces (abstraction layers) to the physical agents: Human Machine ITF and Automated Agent Execution ITF. This leads to the architecture of Figure 47.

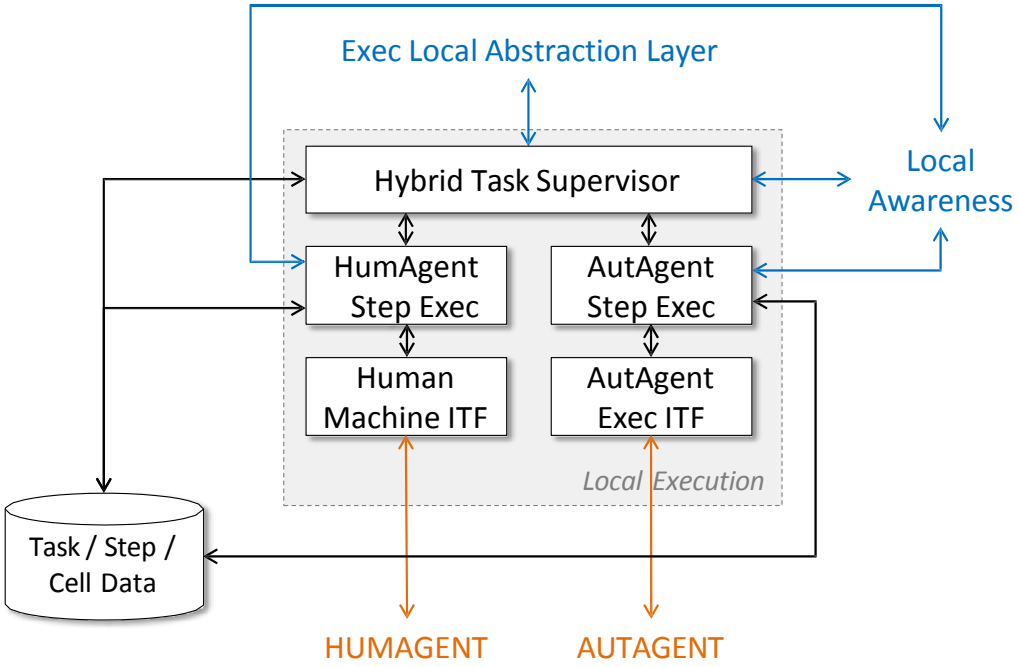

Figure 47: logical software architecture of local execution module

14.2 Local awareness module

The software architecture of the Local Awareness module is shown in Figure 48.

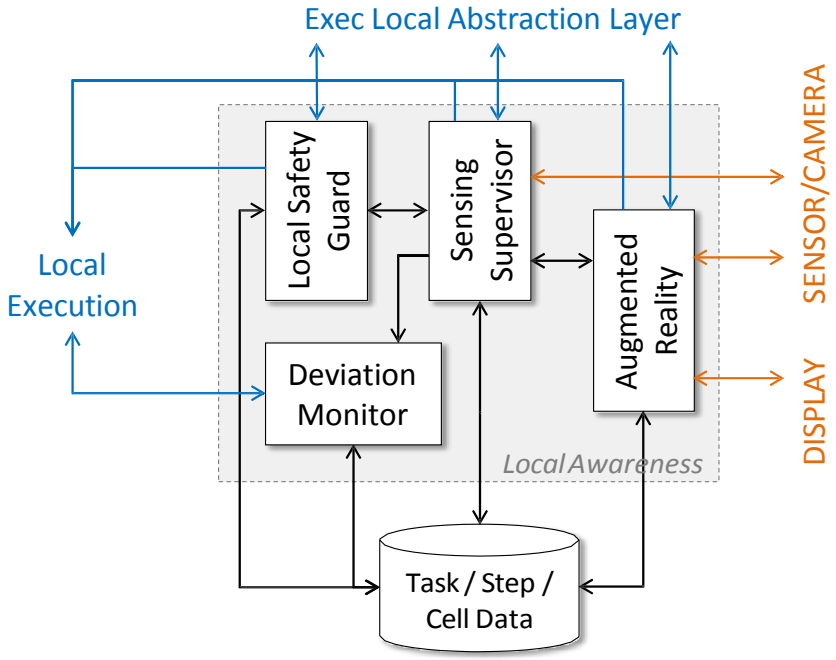

Figure 48: logical software architecture of local awareness module

14.3 HORSE Exec Local overview (aggregation level 4)

Figure 49 shows the overview of the HORSE Exec Local logical architecture (i.e., the combination of Figure 47 and Figure 48). To improve readability of the figure, we have dotted the connection between software modules and the database.

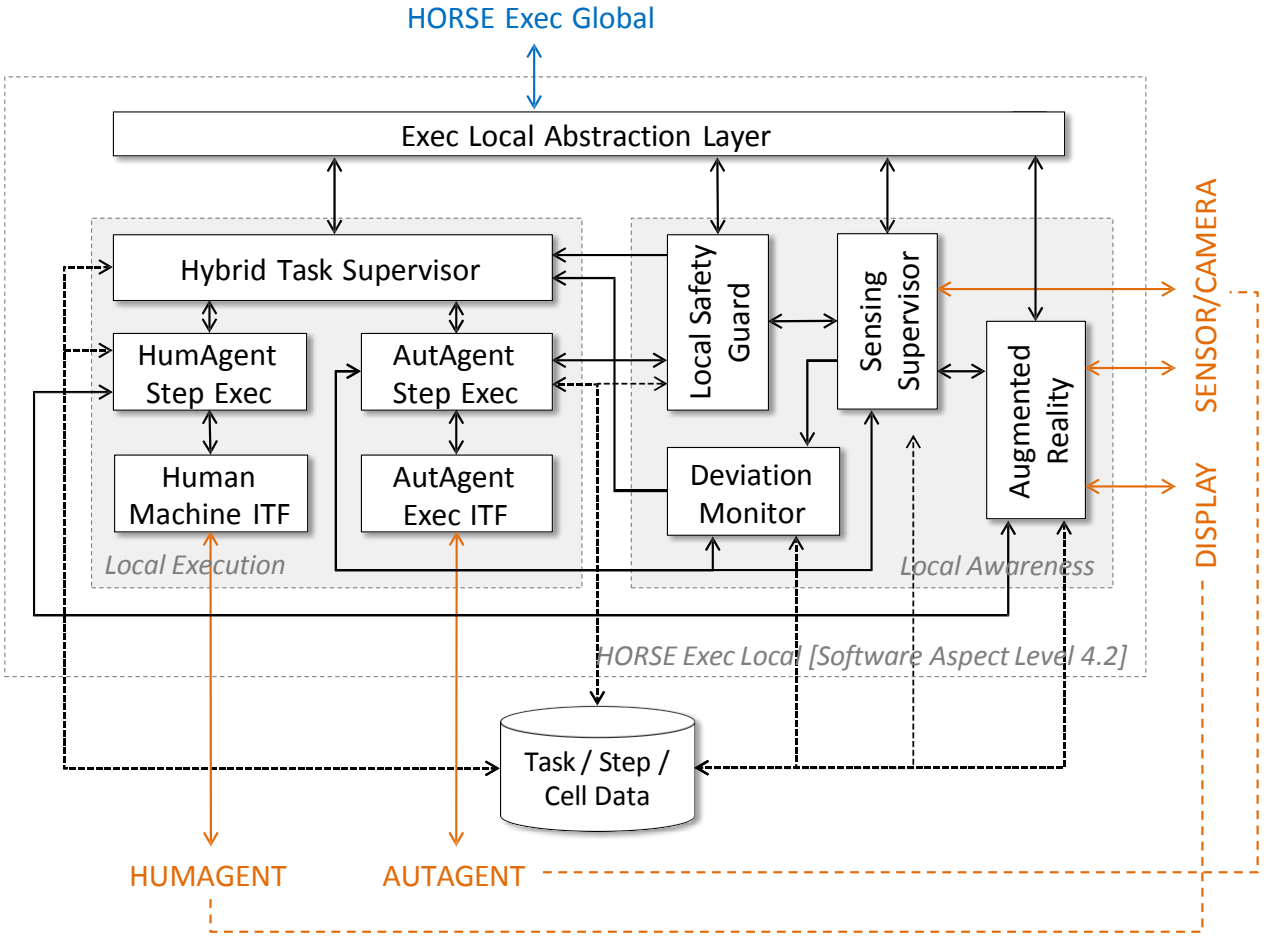

Figure 49: HORSE Exec Local logical architecture, aggregation level 4

As shown in the figure, the Local Execution module is connected to the Local Awareness module via several interfaces. The Hybrid Task Supervisor is connected to the Local Safety Guard and Deviation Monitor, as it has to decide about synchronization of agents in case of security issues (for example, a robotic agent comes too close to a human agent in the execution of a manufacturing task) and observed deviations. The Automated Agent Step Execution module is connected to the Local Safety Guard and Sensing Supervisor to be able to executed manufacturing steps in a reliable and safe way. The Automated Agent execution Interface module is connected to the deviation monitor to pass robot movement signals to the Local Awareness module.

The HORSE Exec Local subsystem interacts with a number of physical resource classes: human agents, automated agents, sensors and cameras, displays. Instances of these classes can be connected in the physical world (as indicated by the orange dotted lines in Figure 49). For example,

a display may be worn by a human agent or a camera can be attached to a robotic agent. These *attached-to* physical relations are not taken into account in the logical architecture.

HORSE
Complete System Design
Part 4:

Conclusion, Bibliography and Appendices

15 Conclusions

This document presents the complete design of the HORSE system, focusing on the logical architecture of the K4+1 framework and using the aspects of the UT5 framework.

The design is based on a number of explicit HAT decisions, which are clearly marked in this document.

The design implies a number of recommendations with respect to the hand-over of the logical system design to the development system design in WP3. These hand-over recommendations are clearly marked in this document.

16 References

[Fowl03] M. Fowler; *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)*; Addison-Wesley Professional, 2003.

[Gref98] P. Grefen, R. Remmerts de Vries; *A Reference Architecture for Workflow Management Systems*; Data & Knowledge Engineering; Vol. 27, No. 1; North Holland - Elsevier; 1998; pp. 31-57.

[Gref15] P. Grefen; *Business Information System Architecture (Version Spring 2015)*; Eindhoven University of Technology, 2015.

[GTI14] *Industrie 4.0: Smart Manufacturing for the Future*; Germany Trade & Invest, 2014.

[Heyw16] R. Heywood (ed.); *UML Use Case Diagrams: Tips and FAQ*; https://www.andrew.cmu.edu/course/90-754/umlucdfaq.html; inspected 2016.

[Hill07] R.Hilliard; *All About IEEE Std 1471*; 2007; available at http://www.iso-architecture.org/ieee-1471/docs/all-about-ieee-1471.pdf.

[HOR16] *System Requirements Specification*; HORSE Project Deliverable 2.1; HORSE Consortium, 2016 (restricted availability).

[IEC13] *Enterprise-Control System Integration - Part 1: Models and Terminology, 2nd ed.*; The International Electrotechnical Commission (IEC), Geneva, Switzerland, 2013.

[Kruc95] P. Kruchten; *Architectural Blueprints—The "4+1" View Model of Software Architecture*; IEEE Software, Vol. 12, No. 6; IEEE, 1995, pp. 42-50.

[Port85] M. Porter; *Competitive Advantage: Creating and Sustaining Superior Performance*; Free Press, 1985.

[Shaw96] M. Shaw, D. Garlan; *Software Architecture: Perspectives on an Emerging Discipline*; Prentice Hall, 1996.

[Truy90] J. Truijens, A. Oosterhaven, R. Maes, H. Jägers, F. van Iersel; *Informatie-infrastructuur: een Instrument voor het Management*; Kluwer Bedrijfs-wetenschappen, 1990 (in Dutch).

[Wik16a] *Class Diagrams*; Wikipedia: https://en.wikipedia.org/wiki/Class_diagram; inspected 2016.

17 Appendix A: Data flow analysis interfaces (Level 2)

In this appendix, we present the data flow analysis on the interfaces of the software architecture at aggregation level 2, as shown in Figure 18 and Table 1. The purpose of this data flow analysis is to check completeness and consistency of the interface listing.

The analysis is presented below in Figure 50 and Figure 51. the interface IDs are those listed in Table 1.

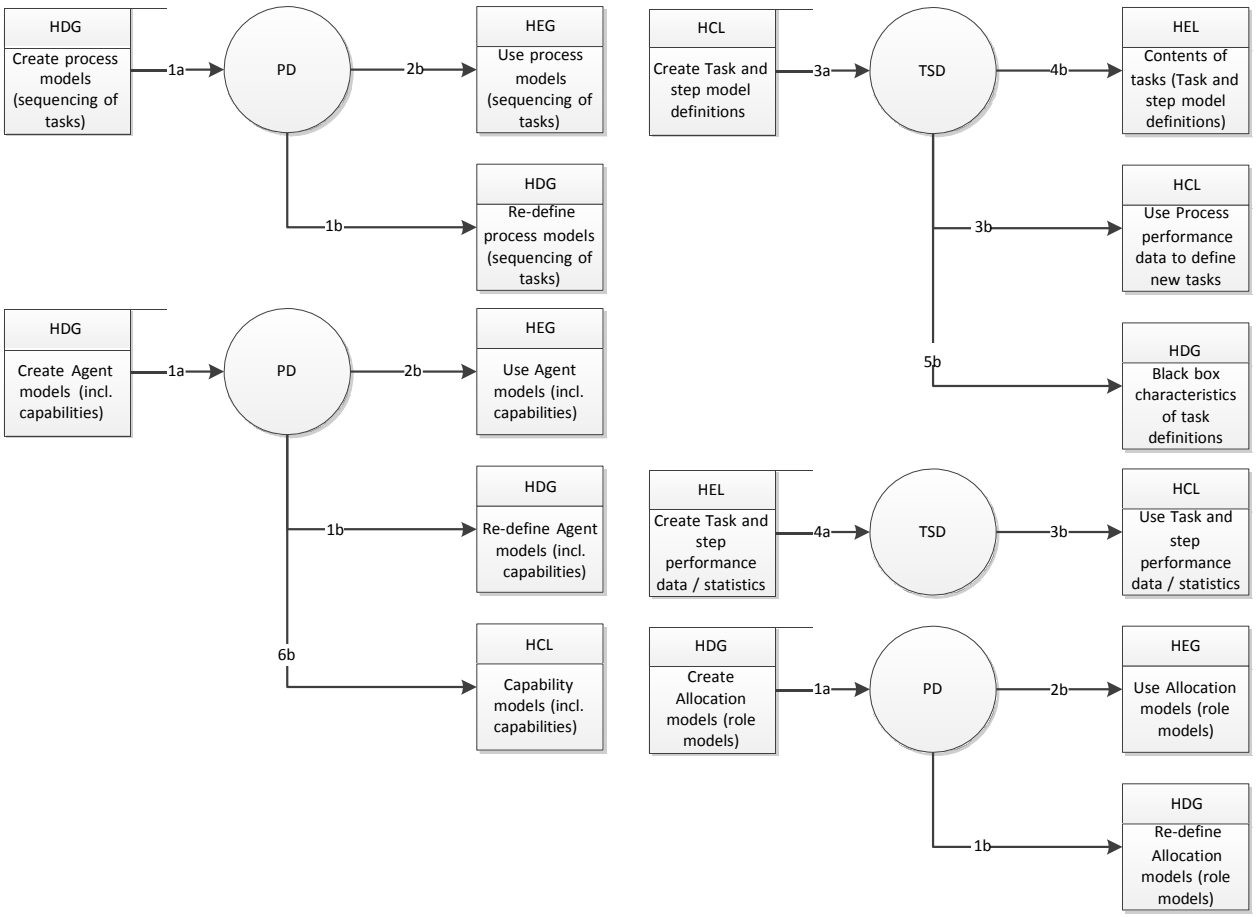

Figure 50: Data flow analysis interfaces software architecture aggregation level 2 (part 1)

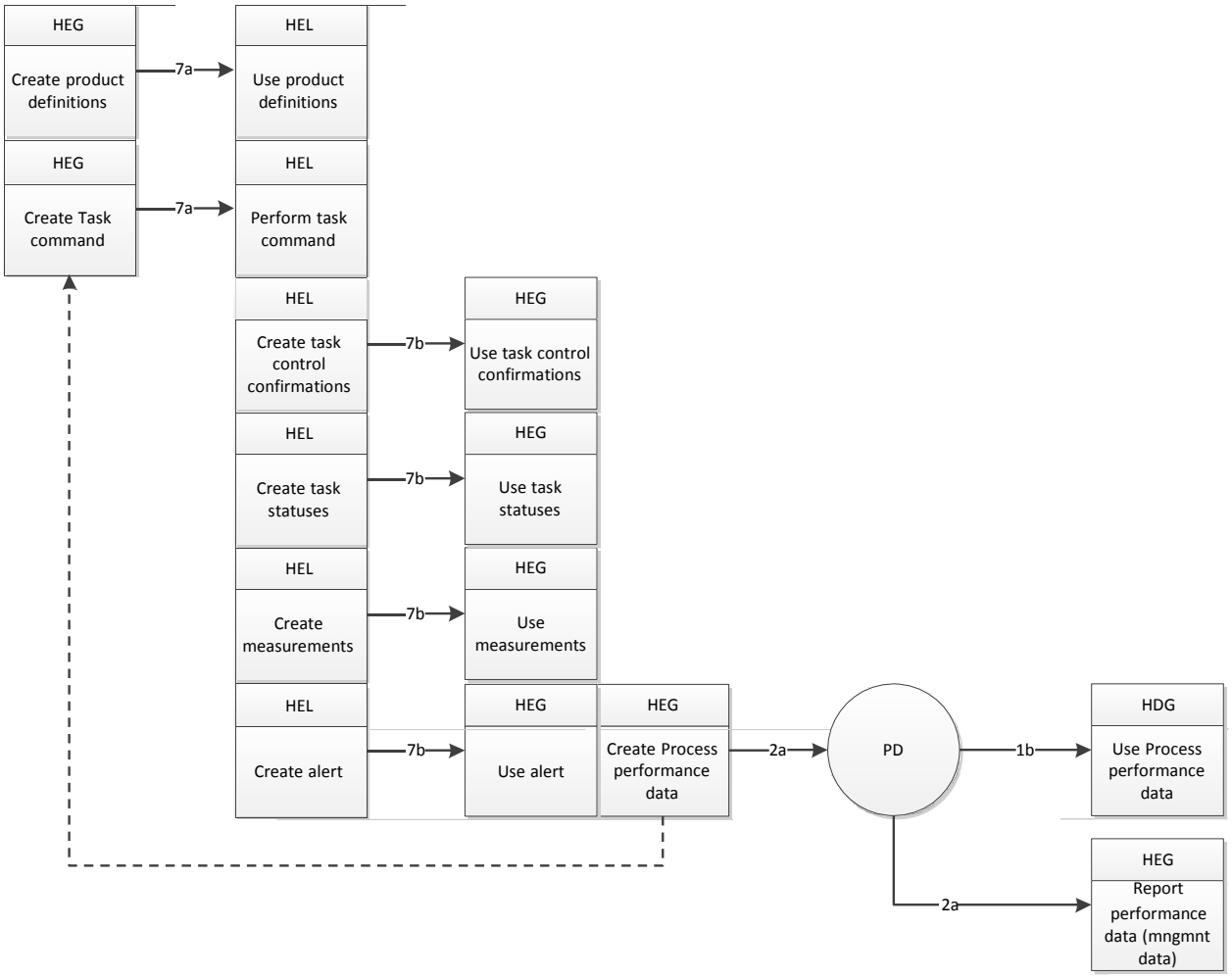

Figure 51: Data flow analysis interfaces software architecture aggregation level 2 (part 2)

18 Appendix B: Database refinement

In this appendix, we discuss a possible refinement of the databases used in the logical software architecture. We first discuss the general database structure. Next, we discuss how specific databases used in HORSE Exec Local at aggregation level 5 map to this general structure.

18.1 General database structure

As a basis, we take the five databases shown in Figure 16 (logical software architecture, aggregation level 2, separated databases. We then apply the distinction made in the Mercurius reference architecture [Gref98]. This leads to the refinement shown in Figure 52. The databases presented in blue are external to the HORSE system, i.e., they reside in the systems in the context of the HORSE system as shown in Figure 11.

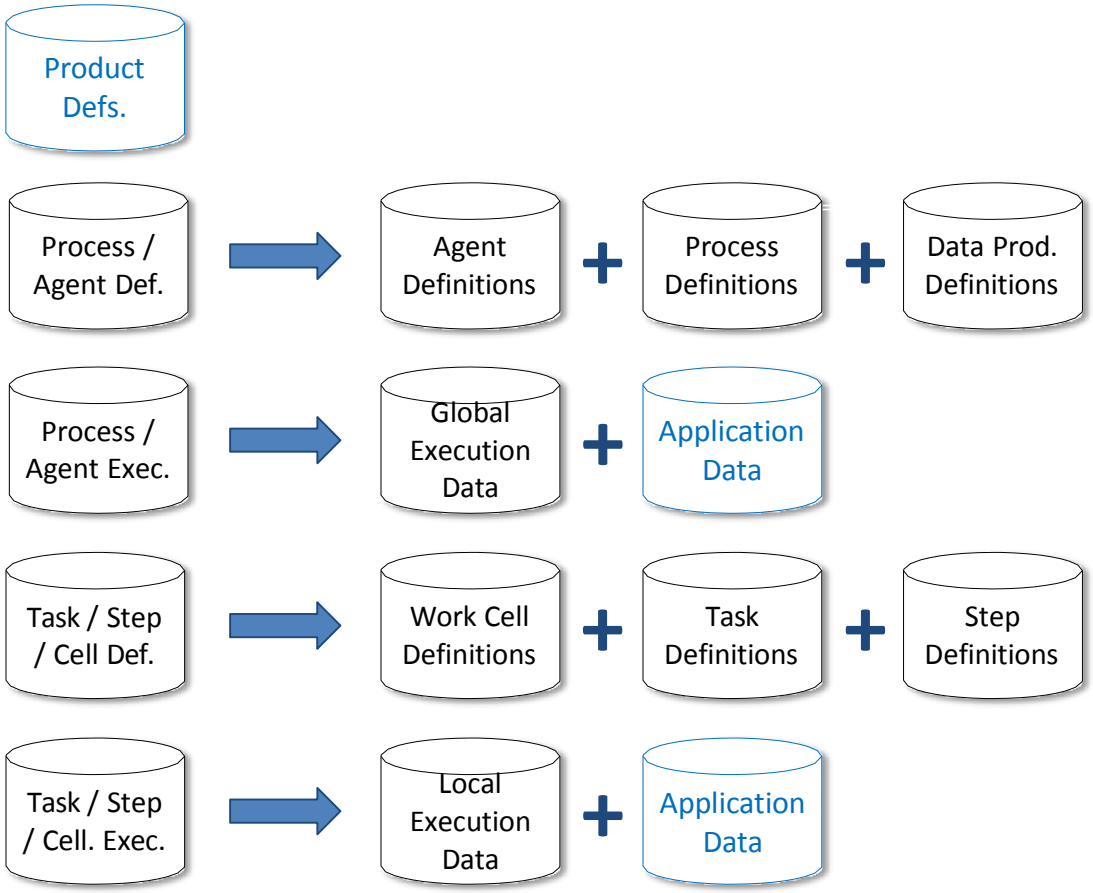

Figure 52: refinement of databases based on WFM reference architecture

18.2 HORSE Exec Local database structure

In Section **Error! Reference source not found.**, we have discussed the HORSE Exec Local logical software architecture. This architecture uses a set of databases from the local execution point of

view. These databases can easily be mapped to the general set of databases discussed before in this appendix. We show this mapping in Figure 53.

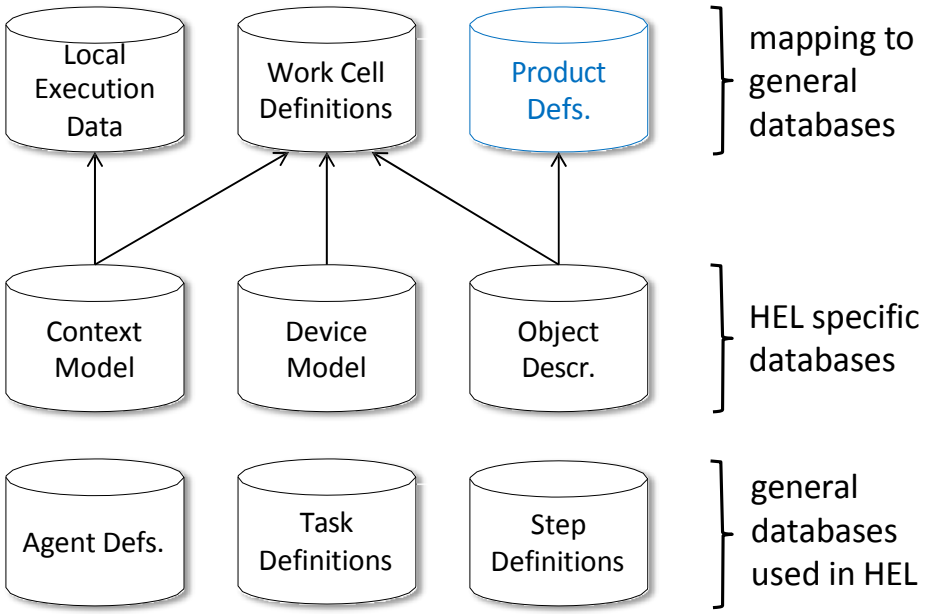

Figure 53: mapping between general HORSE databases and HEL databases

The following remarks hold for the mapping:

- The Context Model database describes the context of work cell tasks being executed. This includes both static data (in the Work Cell Definitions database) and dynamic data (in the Local Execution Data database).
- The Device Model database describes the devices involved in the execution of manufacturing tasks. In the context of HORSE, this is a subset of the Work Cell Definitions database.
- The Object Descriptions database describes the objects relevant in the execution of manufacturing tasks. This includes both objects being manufactured (in the Products Definitions database) and objects explicitly present in the manufacturing environments (such as obstacles impairing robot movement, in the Work Cell Definitions database). This implies that objects not explicitly present in work cells are not considered.
- The Step Defintions database contains Augmented Reality definition data.

19 Appendix C: Hierarchical software component list

This appendix contains a hierarchic component list of the HORSE logical software architecture. For easy referencing in software development in WP3, all components are given a hierarchical software module ID.

19.1 Aggregation levels 2-4, full logical software architecture

Table 4 contains a full enumeration of all components at aggregation levels 2 to 4. We omit level 1 as this level only distinguishes between the global and local levels, which is also implied by level 2.

Aggregation Level 2	Aggregation Level 3	Aggregation Level 4
HORSE Design Global SM1	Process Design SM1.1	Process Flow Modelling SM1.1.1
		Syntax Violation Detection SM1.1.2
		Process Animation SM1.1.3
		Task Identification SM1.1.4
		Agent Class Allocation SM1.1.5
	Agent Design SM1.2	Human Agent Design SM1.2.1
		Automated Agent Design SM1.2.2
HORSE Exec Global SM2	Global Execution SM2.1	Production Execution Control SM2.1.1
		Next Task Selection SM2.1.2
		Agent Selection SM2.1.3
		Worklist Delivery SM2.1.4
		Production Execution Monitoring SM2.1.5
		Structured Exception Handling SM2.1.6

		Global Performance Tracking SM2.1.7
	Global Awareness SM2.2	Global Safety Guard SM2.2.1
		Event Processing SM2.2.2
	Exec Global Abstration Layer SM2.3	*no refinement*
HORSE Config Local SM3	Task Design SM3.1	Task Design Interface SM3.1.1
		Task Parser SM3.1.2
	Human Step Design SM3.2	Human Step Design Interface SM3.2.1
		AR Design SM3.2.2
	Automated Step Design SM3.3	Automated Step Design Interface SM3.3.1
		Automated Step Planner SM3.3.2
	Workcell Design SM3.4	Workcell Simulator SM3.4.1
		Workcell Configuration SM3.4.2
HORSE Exec Local SM4	Local Execution SM4.1	Hybrid Task Supervisor SM4.1.1
		Human Agent Step Execution SM4.1.2
		Human Machine Interface SM4.1.3
		Automated Agent Step Execution SM4.1.4
		Automated Agent Execution Interface SM4.1.5

	Local Awareness SM4.2	Local Safety Guard SM4.2.1
		Sensing Supervisor SM4.2.2
		Deviation Monitor SM4.2.3
		Augmented Reality SM4.2.4
	Exec Local Abstraction Layer SM4.3	*no refinement*

Table 4: hierachical component list logical software architecture levels 2-4