TOWARDS AN ONTOLOGY FRAMEWORK FOR THE INTEGRATED DESIGN OF MODULAR ASSEMBLY SYSTEMS

Niels Lohse, MSc., Dipl.-Ing. (FH)

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

May 2006

To my lovely, lovely wife Eleftheria,

who supported me with immeasurable patience

Abstract

Next generation manufacturing companies have to become highly responsive in order to succeed in an ever more rapidly changing global market. The ability to effectively develop and adapt their assembly facilities (systems) to changing requirements on demand plays a crucial role in achieving high responsiveness since the assembly process has to deal with the full inherent complexity of increasingly mass-customised products.

This work was motivated by the current lack of a holistic assembly system design theory that would enable design environments to address the need for rapid system development and adaptation. The challenge is to create a common environment where domain experts can effectively collaborate while taking advantage of the best practices of their diverse domains.

This thesis investigates how a domain ontology can help to overcome those challenges. The approach is taking advantage of the higher levels of standardisation inherent in the modular assembly system paradigm which is considered to be one of the fundamental enabling factors to achieve a high level of adaptation.

A new ontology framework has been developed to support the design and adaptation of modular assembly systems (ONTOMAS). The ONTOMAS framework is based on engineering ontology principles structuring the domain using formalisms for aggregation, topology, taxonomies, and system theory principles.

A number of design patterns have been identified and formalised to support key design decision-making tasks during the design of modular assembly systems. Furthermore, the function-behaviour-structure paradigm has been applied to capture the characteristics of modular assembly equipment at different levels of abstraction that reflect the specific needs of the engineering design process.

The proposed ONTOMAS framework provides a sound foundation for computer based support tools to reduce the assembly system design effort and time while maintaining a high level of quality. An integrated design framework for the requirements driven specification of assembly processes and configuration of modular assembly system has been developed. The design approach applies the new formalisms of ONTOMAS to support the design decision-making activities.

Abstract

The developed ONTOMAS framework has been applied in several industrial and synthetic use cases to verify its applicability and appropriateness. Furthermore, the new ontology and design framework have been used as foundation for the development of a prototype collaborative design environment which allows different domain experts to participate in the design of modular assembly systems.

List of Publications

Journal Publications:

Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, *"Equipment Ontology for Modular Reconfigurable Assembly Systems"*, accepted for publication in the International Journal of Flexible Manufacturing Systems, Springer, ISSN 1572-9370

Lohse, Niels, Ratchev, Svetan, Valtchanov, George (2004), *"Towards Web-Enabled Design of Modular Assembly Systems"*, Assembly Automation, Emerald, Vol. 24, No. 3, pp. 270-279

Ratchev, Svetan, Lohse, Niels (2004), *"Data modelling for web enabled design of modular precision assembly devices"*, Assembly Automation, Emerald, Vol. 24, No. 1, pp. 63-70

Bock Sections:

Lohse, Niels, Schäfer, Christian, Ratchev, Svetan, (2006), *"Towards an Integrated Assembly Process Decomposition and Modular Equipment Configuration"*, Precision Assembly Technologies for Mini and Micro Products, edited by: Ratchev, S., Springer, New York, ISBN 0-387-31276-5

Fully Refereed Conference Papers:

Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, (2005), *"Equipment Ontology for Modular Reconfigurable Assembly Systems"*, Proceedings of the CIRP 3rd International Conference on Reconfigurable Manufacturing, 10-12 May 2005, Ann Arbor, MI, USA

Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, (2005), *"An Ontology for the Definition and Validation of Assembly Processes for Evolvable Assembly Systems"*, Proceedings of the IEEE International Symposium on Assembly and Task Planning, 19-21 July 2005, Montreal, Canada

Ratchev, Svetan, Lohse, Niels, Hirani, Hitendra (2004), *"Knowledge model for distributed conceptual design of modular assembly workstations"*, Proceedings of the 35th International Symposium on Robotics, 23-26 March 2004, Paris-Nord Villepinte, France

Lohse, Niels, Ratchev, Svetan, Chrisp, Anthony (2004), *"Function-Behaviour-Structure model for modular assembly equipment"*, Proceedings of the International Precision Assembly Seminar 2004, 11-13 February 2004, Bad Hofgastein, Austria, pp. 167-174

Hirani, Hitendra, Ratchev, Svetan, Niels, Lohse, Valtchanov, George (2004), *"Web-based specification of reconfigurable precision assembly system – industrial scenarios and use cases"*, Proceedings of the International Precision Assembly Seminar 2004, 11-13 February 2004, Bad Hofgastein, Austria, pp. 27-34

Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan (2003), *"Task-Based Modelling and Configuration of Assembly Workstations"*, Proceedings of the IEEE International Symposium on Assembly and Task Planning 2003, Besançon, France, 9-11 July 2003, pp 301-306

Ratchev, Svetan, Lohse, Niels (2003), *"Data Modelling for web enabled design of modular precision assembly devices"*, Proceedings of the International Precision Assembly Seminar 2003, 17-19 March 2003, Bad Hofgastein, Austria, pp. 149-156

Ratchev, Svetan, Lohse, Niels (2002), *"Linguistic based configuration of modular precision assembly devices"*, Proceedings of 33rd International Symposium on Robotics 2002, Stockholm, Sweden, 7-11 October 2002, pp. 557-562

Acknowledgements

I would like to thank a number of individuals for their support and contributions without which this work would not have been possible. First of all I would like to especially thank my supervisor Svetan Ratchev for having given me the opportunity to embark on this adventure and express my gratitude for his support and constant belief in my work.

I would also like to thank my colleagues in the Precision Manufacturing Group at the University of Nottingham. Special thanks go to Hitendra Hirani and George Valtchanov whose comments and support were invaluable for this work.

Furthermore, I wish to express my appreciation to all the people I have worked with on AssemblyNet, E-Race and EUPASS projects for their comments and feedback. This, I believe, has strongly contributed to the quality and soundness of the work. I would like to also especially recognise the support and encouragement I received from José Barata (Universidade Nova de Lisboa), Mauro Onori (KTH Stockholm), and Mark Jones (TQC Ltd, Nottingham).

A big thank you also goes to the people that I worked with at Bosch Corporate Research. I would especially like to express my gratitude to Christian Schäfer for giving me the opportunity to join his team and for all the inspiring discussions we had together. Furthermore, I would like to thank Zhiliang Qi for his feedback on my work and his insight of the subject domain.

I wish to express my warmest thank you to my parents Olaf and Angelika Lohse for their unfailing support and encouragement. Without their support I would have never even been in a position to consider the PhD degree as an option.

Finally, but most importantly, I would like to thank my wife Eleftheria Lohse who through her belief, patience and love, has given me the will, determination and strength to bring this work together.

Table of Contents

Table of Figures

Symbology

Liaison

Liaison Capabilities (Type/Individual)

Actual Liaison (Occurrence)

Liaison Requirements

Liaison Class (Concept)

Liaison Constraint (Pattern)

Component/Assembly

Activity (Process)

Activity Capabilities (Type/Individual)

Actual Activity (Occurrence)

Activity Requirements

Activity Variant (Sequence)

Activity Class (Concept)

Activity Constraint (Pattern)

Function

Function Capabilities (Type/Individual)

Actual Function (Occurrence)

Conceptual Function (Requirements)

Equipment Function View

Behaviour

Behaviour Capabilities (Type/Individual)

Actual Behaviour (Occurrence)

Conceptual Behaviour (Requirements)

Equipment Behaviour View

Equipment

Equipment Capabilities (Type/Individual)

Actual Equipment (Occurrence)

Conceptual Equipment (Requirements)

Equipment Variant/Equipment Structure View

Equipment Class (Concept)

Equipment Constraint (Pattern, Module?)

State

Male Port (Output)

Male Component Port

Female Port (Input)

Female Component Port

Neutral Port

<isA> Generalisation Relationship

<instanceOf> Instantiation Relationship

<occurrenceOf> Occurrence Relationship

Attribute Relationship

Type Relationship

<hasParts> Aggregation Relationship

1 Introduction

Next generation manufacturing companies have to become highly responsive in order to succeed in an ever more rapidly changing global environment (NGM Project [89]). Some major imperatives have been identified to meet this challenge for today's companies. They include: workforce flexibility; knowledge supply chains; rapid product/process realisation; innovation management; change management; next-generation manufacturing processes and equipment; pervasive modelling and simulation; adaptive, responsive information systems; extended enterprise collaboration; and enterprise integration.

The Integrated Manufacturing Technology Initiative (IMTI Report [58]) has identified the following "Grand Challenges" for manufacturing success in the 21st century: Lean, Efficient Enterprises; Customer-Responsive Enterprises; Totally Connected Enterprises; Environmental Sustainability; Knowledge Management; and Technology Exploitation.

Assembly is one of the key focus areas in manufacturing especially since, with the increasing demand for mass customised products, the assembly process has to cope with the full inherent range of product variety. Ever more demanding market requirements in a number of key industrial sectors such as telecommunication systems, precision medical equipment and electronics, dictate the necessity to continuously increase the functional density in their products. Consequentially part sizes are declining and precision assembly becomes one of the key factors. Another tendency especially in the consumer product industry is the constantly declining product lifetime whereas at the same time the required investments for assembly solutions are not declining respectively; on the contrary they are rising due to increasing complexity.

The Reconfigurable and Evolvable Assembly Systems paradigm is aimed to address the needs of next generation manufacturing systems by enabling enterprises to rapidly respond to changes in today's increasingly volatile and dynamic global markets (Koren, et al. [64] and Onori, et al. [91]). The objective is to overcome the need for substantial investment costs into excess flexibility that may or may not be required to react to changes in the future. One of the key challenges of creating

effective evolvable systems is to reduce their adaptation effort and ensure continuous improvement. Modular system architectures are considered to be one of the enabling factors to address this challenge. Modular assembly systems on their own, however, only provide the technical capabilities but do not address the need for a purpose driven, optimised adaptation. The required iterations, especially on the structural level, cannot take place in the operational system since the resulting down times from reconfigurations and trials would be prohibitive. This limitation can be overcome by integrating the modular system paradigm with synthetic design tools that translate the changing user requirements into best possible system solutions, using state-of-the-art simulation methods and knowledge enabled optimisation techniques before triggering a physical system reconfiguration.

A significant research effort has already been directed towards the addressing the challenges of rapidly reconfigurable manufacturing solutions (RMS Center [106], EUPASS [33], AAA/Minifactory [1]). Despite the considerable research in the area there is still a strong need to investigate further the role of design tools and enabling knowledge ontologies in the evolvable assembly system paradigm.

This work is investigating how design tools and their supporting knowledge ontologies can enable modular assembly system to become rapidly reconfigurable and evolvable. The work is addressing the need for rapid product/process realisation, knowledge enabled enterprises, and intelligent design tools as identified by the roadmaps in the domain (NGM Project [89], IMTI Report [58], Onori, et al. [91]). The ultimate motivation is to reduce the effort and time for design and integration of modular assembly systems while maintaining and improving their quality.

The aim is to create a rapid assembly system development method based on capability matching of modular equipment solutions. The fundamental notion (hypothesis) is that a high percentage of the assembly process requirements within any one specific industry sector can be covered with a finite set of standardised assembly equipment modules. This widens the scope for the definition of a highly automated configuration method for modular assembly systems based on assembly process requirements, which would allow assembly system designers to focus on the critical, new or unstable design problems.

1.1 Motivation

Three main stakeholders normally participate in the design of assembly systems: customers who need an assembly system to assemble their products, system integrators who provide the capability to develop assembly system solutions, and equipment suppliers who design and supply the fundamental building blocks of an assembly system. The role of the system integrator is of primary importance for the reported investigation. Their current design practice is outlined here to illustrate the need for more advanced methods and tools during the design and integration of assembly systems.

It is the role of the System Integrator to integrate functional equipment components into an overall system that fulfils a given set of user requirements. In the case of assembly system integration those user requirements are generally defined around a product that needs to be assembled. They include definitions of how the parts of the product need to be put together, operational constraints, constraints for the production system, and project management related aspects. The part relationships are the most important aspects of the user requirements for the technical realisation of the assembly system. They determine the required assembly processes which constitute the system requirements for the design of an assembly system. The types of part relationships define the required processes and the geometric topology of the product constraints the order in which the processes can take place.

The definition of the system requirements falls within the responsibility of the system integrator. They extract them in the first instance from the given set of user requirements. It is often possible that more than one set of system requirements can fulfil the given user requirements. The major variation is introduced through different possible process orders which define the core aspects of the system requirements. The system integrator has to decide, based on his experience, which approach is most promising in terms of implied system cost. The decision is often based on past experience from similar projects. This decision can have a significant impact on the design cost of the assembly system since later iterations will incur heavier design effort penalties. The major driver for cost is the type and characteristics of the required assembly processes. These, however, are often fixed through the user requirements and can only be changed through negotiations with the user/product designer.

The conceptual design of the assembly system starts once the overall process order has been defined. This normally entails a further detailing of the process definition to reach a level that is closer related to the actual functional capabilities of physical equipment solutions that will make up the assembly system. This process is also called function analysis. The focus at this point is on defining possible conceptual structures for the needed system which assigns required process steps to appropriate equipment types. It also includes a grouping of process steps into stations and cells. Again the key decision factor at this point is the experience of the system integrator who needs to understand the right equipment types and how to group them together to achieve a balanced system. The decision criterion at this point is normally the cycle time that needs to be achieved and the overall cost of the system. It is the responsibility of the system integrator to judge how long the individual process steps are likely to take. They are normally estimated from past experience on similar projects.

Once the conceptual design has been defined to a sufficient level of detail the embodiment design starts with the selection of key functional equipment components. They are normally selected from equipment supplier catalogues taking the experience and available expertise of the system integrator into consideration. Some equipment components are subcontracted to be custom designed by outside equipment design specialists. This is very common, for example for the customisation of part feeders that need to be adjusted to the specific needs of the individual product parts. The main responsibility of the system integrator during the embodiment design is to combine all the different functional equipment components into one working assembly system. This includes the design of the mechanical structure, electrical and pneumatic wiring, and custom control and software development.

During the design process it is the responsibility of the system integrator to demonstrate and convince the end user that their proposed design fulfils their technical needs and that it is going to be to their economic advantage. This interaction and negotiation with the end user takes place quite early during the design of the system and is normally based on conceptual definitions. This is part of the bidding process. Normally more than one system integrator is asked by an end user to provide quotations for assembly system solutions. It is important for a system integrator to demonstrate technically sound solutions and simultaneously minimise the overall cost. It is important for a system integrator to be able to provide balanced quotations at an

early design stage. Since they are generally based on conceptual design specifications it is critical for the system integrator to be able to make realistic estimates of their required design and equipment costs. One of the critical factors for a system integrator to gain a competitive advantage is their ability to reduce integration and design effort.

Integrated and knowledge enabled methodologies are one way to reduce the design and integration effort by making the right information available at the right time and by providing state-of-the-art engineering tools that support the development.

Another approach is to increase the reuse and repetition of system component. Most system integrators have developed their own system architectures and system design approaches to help them reduce design and integration effort. These architectures vary from system integrator to system integrator. Some have defined highly standardised modular approaches that focus on delivering the most common functional capabilities within a targeted domain. Others are more concerned with maintaining a wider range of possible system solutions and have created less rigid more abstract guidelines for their engineers. Today the main effort of creating system architectures and integrating existing equipment components lies with the system integrator. They need to design suitable mechanical frameworks, select the right functional equipment components that cater for a wide range of user requirements, and adapt them to fit into their structure. In the future this could change by creating consortia of customers, system integrators, and equipment suppliers wich define domain wide system architectures that are mutually beneficial for all of them and could significantly reduce their development effort. The EUPASS project is currently aiming to achieve this objective (EUPASS [33]).

The implication for supporting design frameworks and knowledge ontologies is that they should provide mechanisms that take advantage of this higher level of standardisation. This opens up the scope and need for a higher degree of integration and automation during the design of such systems. Configuration methodologies that have been demonstrated in the computer industry which benefit from a higher degree of modularization can be harnessed to solve the challenges of the assembly system design process. Examples of such configuration methods include XCON (McDermott [79]), MICON (Birmingham, et al. [8]), and COSSACK (Mittal and Frayman [83]).

1.2 Aim and Objectives

The aim of this work is to ***define a suitable design framework and supporting ontologies that enable rapid design of modular assembly systems***. The objectives include the definition of a suitable design framework, the definition of the required domain concepts and their interrelationships, as well as the definition of suitable knowledge support formalisms to guide and support the design process. The following more detailed objectives for the reported work have been identified:

- Definition of a new assembly process model which allows the specification of the required process capabilities at a level of detail that is sufficient for the selection of sub-workstation assembly equipment modules. The definition should focus on the following aspects to fulfil the needs for the assembly process definition:
    - Dynamic definition of the interrelationships between the individual assembly process steps that is suitable for iterative and concurrent design approaches.
    - The intended meaning of the process specification needs to be interpretable by computer based reasoning applications to maximise the design automation.
    - The proposed model needs to be able to deal with the inherent complexity of the required high levels of detail
    - The model should be integrated into the wider domain framework and maintain constraints to the product and equipment definition.
- Development of a new assembly equipment model that enables the process-requirements-driven selection and integration of modular assembly workstations. The development should focus on the following aspects to achieve the desired objective:
    - The equipment model needs to capture the specific constraints of the modular assembly paradigm to allow a seamless integration of equipment module descriptions into a wider system solution.
    - The description of the equipment modules should support their assembly process requirements based selection, integration, and evaluation.

- The intended characteristics of the equipment specification on both individual as well as composite level should be accessible for computer interpretation
- The proposed model needs to be able to deal with the high level of complexity inherent in detailed equipment specifications
- It should be possible to integrate the equipment model into the wider design space to enable dynamic maintenance of domain wide design constraints

- Formulation of a new method that can integrate the process specification and assembly workstation configuration using the developed process and equipment models. The formulation of the design approach should focus on the following aspects to enable the integration:
  - Dynamic decomposition and specialisation of assembly tasks
  - Enable the matching of required process capabilities against existing hardware capabilities
  - Dynamic integration of equipment modules into assembly workstation
  - Maintenance of design constraints across the whole modular assembly system design domain

1.3 Approach and Structure of the Thesis

Based on the analysis of existing research and current design practice in the area of assembly systems, a new ontology based framework has been defined to support the design of modular assembly systems. The new domain ontologies are split into the three domains that traditionally exist within the area of assembly system design (Rampersad [99]): product domain ontology, assembly process domain ontology, and assembly system/equipment domain ontology.

Figure 1.1 shows the fundamental structure of the thesis. The thesis starts with an analysis of the reported relevant research in the area and identifies currently existing knowledge gaps. The fundamental ideas and assumptions behind the proposed integrated ontological framework are being discussed in the light of the identified knowledge gaps. The three domain ontologies are described and explained in detail in the following three chapters. Their applications and a prototype application are being shown to validate their potential. Finally the thesis concludes with a discussion of the

outcomes of the work, how it contributes to the current body of knowledge, and further work required.

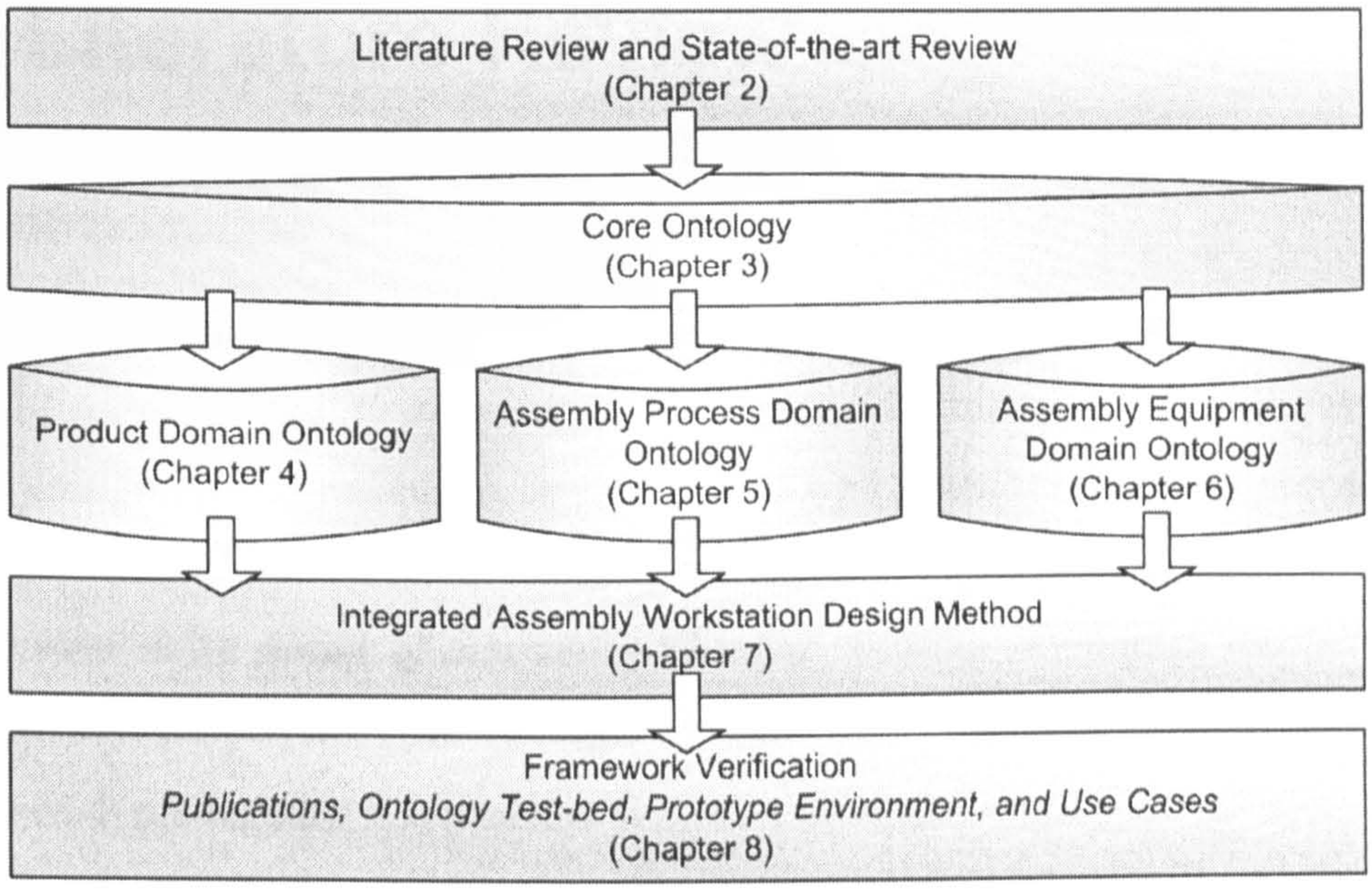


Figure 1.1Thesis Structure Overview

2 Literature Review

2.1 Introduction

Assembly is defined as *"putting together of [components] to make a product"* or *"a set of [components] so assembled"* (OED [90]). In this work the word assembly will be used in both its meanings, as a process as well as the result of this process.

A system is *"a group of interacting elements forming a complex whole"* (OED [90]). An assembly system can therefore be defined as a group of interacting elements composed to put together components to make a product.

From this definition it can be derived that assembly systems involve three distinct aspects: the product that is being assembled, the process of assembling the product and the actual physical system that executes the processes of assembling the product. Rampersad [99] introduces a model that links the variables of these three aspects and creates an integral model for assembly (see Figure 2.1).

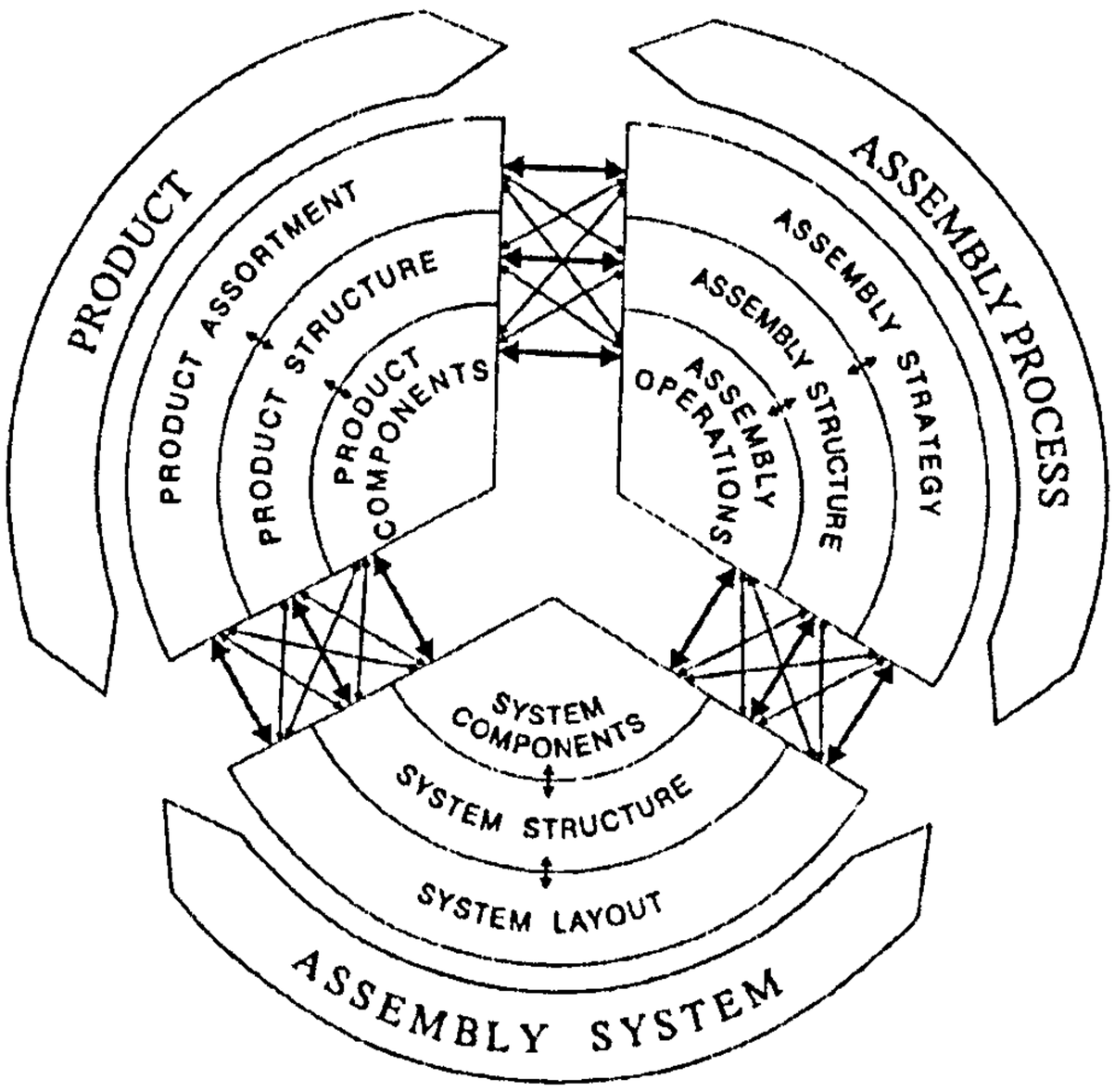


Figure 2.1 Integrated Assembly Model (Rampersad [99])

The product, as has been previously stated, is made up from components. Components can either be piece parts or assemblies of their own, so called sub-assemblies. The components of a product have a relation to one another, the product

structure, which is again part of the overall product assortment of an enterprise. But the product is not only the sum of its parts but rather the means to realise a set of functions, which are based on the needs of its users. There is therefore a causal relation in the form of: users->functions->products->processes->systems.

On the process side the smallest process entity is the assembly of two components, which will be called assembly task. To assemble a product the tasks have to be executed in a defined sequence, which is not only defined by the product but also by the assembly strategy of a company.

The assembly system itself, like the product, is made from a set of system components, which are linked to build the structure of the system. The actual physical system is the dimensional layout of all its components according to its structure and the designated space.

The general design process is a sequence of defining requirements in terms of functions and their relations and linking those functions to actual physical components that either need to be selected from existing ones or newly designed. That means for the design of an assembly system, which its requirements in form of a process description have to be derived from the product. Once the process oriented requirements have been derived from the product description they have to be transformed into an actual assembly system configuration.

Throughout the literature there is a strong link between assembly tasks on the process side and workstations on the assembly system side (Graves and Lamar [44]). It can generally be established that assembly workstations are clearly defined sets of equipment within the assembly system capable of performing a set of one or more assembly tasks. Furthermore, the design of workstations is a distinct sub-problem of assembly system design that can be solved locally and at the same time helps to improve the overall assembly system.

Current assembly system optimisation methods commonly allocate sets of one or more tasks to workstations and select the best suitable combination of workstations to form the assembly system. Available methods, however, do not consider the actual configuration of workstations but rather assume an existing set of workstations that can perform a range of tasks and select the workstations according to constraints like cycle time and evaluation criteria like costs. How these criteria are derived or even improved in a task driven design are generally not considered. This work is focused on closing this gap.

In the following sections the underlying research questions and current research results in the area of modular workstation design and modelling are presented and discussed. The literature review is focused on the three underlying aspects of the work: modularity, design methods, and supporting domain models. The literature review is concluded with an analysis of the knowledge gaps in the currently reported research.

2.2 Background

2.2.1 Modularity

Modularisation of products and systems is maintained to be one of the key strategies to deal with increasing complexity, rapidly changing requirements, and continuous integration of new or improved technologies (Pahl and Beitz [94], Tsukune, et al. [126], Bi and Zhang [7], and Stevens, et al. [118]). Modularisation of systems is also considered to be one of the key enabling factors for next generation agile system solutions like reconfigurable and evolvable assembly systems (Koren, et al. [64], Onori, et al. [91], Hollis and Quaid [53]).

2.2.1.1 Principles and Issues

"Modular products are machines, assemblies and components that fulfil various overall functions through the combination of distinct building blocks or modules." (Pahl and Beitz [94])

From this definition the key ideas of the modular approach can be seen: decomposition of a set of overall functions into a set of distinct lower level functions, which can in turn be combined in different ways to yield any of the original overall functions. Each of these lower level functions, or sub-sets of them, are then associated to physical building blocks, so called modules. Through combination of these building blocks a product/system that exhibits the desired overall function can be synthesised. This reflects the underlying principles of hierarchical system design approaches as will be discussed in more detail in section 2.2.2.

Pahl and Beitz [94] identify two main drivers for product/system modularisation: function-oriented and production-oriented. The function-oriented approach is focused on realising a wide number of overall product/system functions with a small number

of building blocks whereas the production-oriented approach focuses on defining the product components to optimise its overall production effort. The production focus will not be discussed in more detail in this report since the main focus is to deal with function complexity in assembly systems where the assembly system constitutes the product of the design process.

Modularity is an approach to define product/system architectures. In general two types of product/system architecture can be distinguished: *integral* and *modular* (Ulrich [127]). Ulrich and Tung [128] point out that all products have a varying degree of modular and integral architecture depending on the level of abstraction. Integral design of products or components has the advantage that it allows global optimisation of the physical representation.

Ulrich and Tung [128] define the degree of modularity according to the similarity between the physical and functional architecture and the amount of incidental interactions between the physical components.

The degree of similarity between physical components and functions is determined by the number of components that implement a function or by the number of functions a component is required for. An absolute modular structure, therefore, would realise each function with a separate component (one-to-one relation) and on the other extreme an absolute integral architecture would have links between all components and all functions (many-to-many relation).

Incidental interactions between components are all those interactions that are not critical to the function of the product. An ideal modular product/system, therefore, would need to be designed to have no incidental interactions between its components (modules) whereas the components in an integral product/system on the other hand would need to be designed to cope with any incidental interactions with other components.

Hence, modularity defines the relation between functions and components as well as the interactions between components. The interactions between components in a modular architecture are defined through *interfaces*.

Stevens, et al. [118] advocate that *interfaces* between components in a system should be clear, stable and decoupled to establish a truly modular architecture. The clearness and stability aspect of interfaces is required in order to allow different

modules to implement the same interfaces and maintain compatibility. Decoupling is the removal or minimisation of incidental – unintended – interactions through the interface (see above discussion). This allows any two modules that implement the same interface to be connected without any of them having to know how the other one is working internally. Interfaces that are defined in such a manner allow not only the easier integration of modules into a system but also allow the testing of modules outside their application environment. This only requires a test system to provide the desired stimuli through the same interface as the module.

2.2.1.2 Classification of modules and modular systems

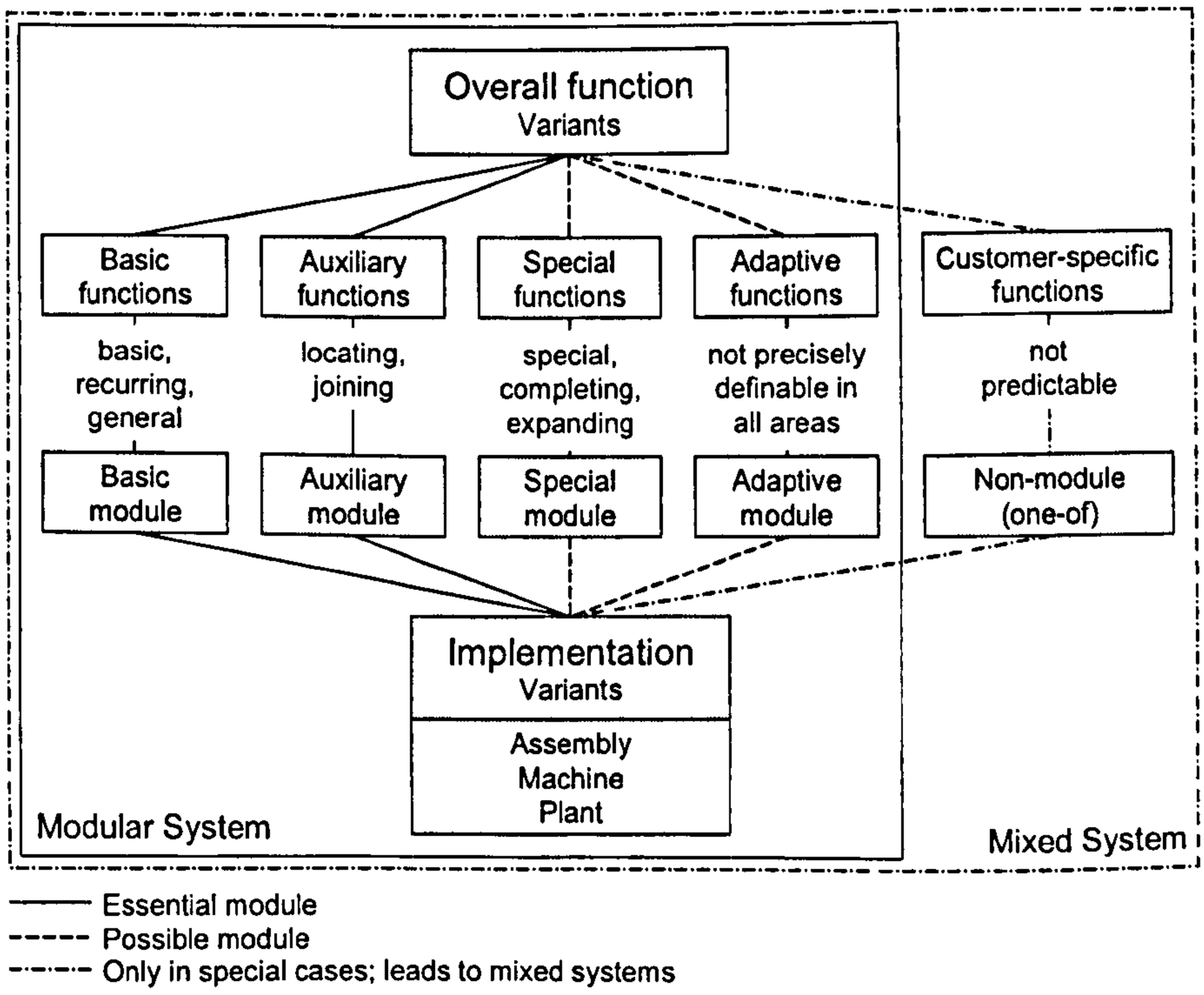


Figure 2.2 Function and module types in modular and mixed systems (Pahl and Beitz [94])

Pahl and Beitz [94] define function module types depending on the function they realise and their importance to the system (essential or possible). The functions are classified according to their role in the overall function of the product/system. They give four types of functions/modules for a modular system: basic, auxiliary, special, and adaptive (see Figure 2.2). Basic functions are the fundamental building blocks for the overall function and appear in all overall function variants. Auxiliary functions are additional or support functions for the basic functions. Special functions are task specific sub-functions and need not appear in all overall function variants. Adaptive

functions are required for adaptation to other systems and to marginal conditions. Customer-specific functions are all those that are not included in the modular system.

Bi and Zhang [7] define a high level taxonomy for the classification of modular *applications* based on the findings of Ulrich and Tung [128], Pahl and Beitz [94] and others. They advocate that any modular application should be defined based on four attributes associated to the components/modules and their interfaces (see Figure 2.3). The type of components/modules of a modular application is classified by the type of component entities and at what level they are in regard to the overall application domain. Interfaces are defined based on how the components are integrated together (component view) and on how the connection between the components is being established by the interface (connection view).

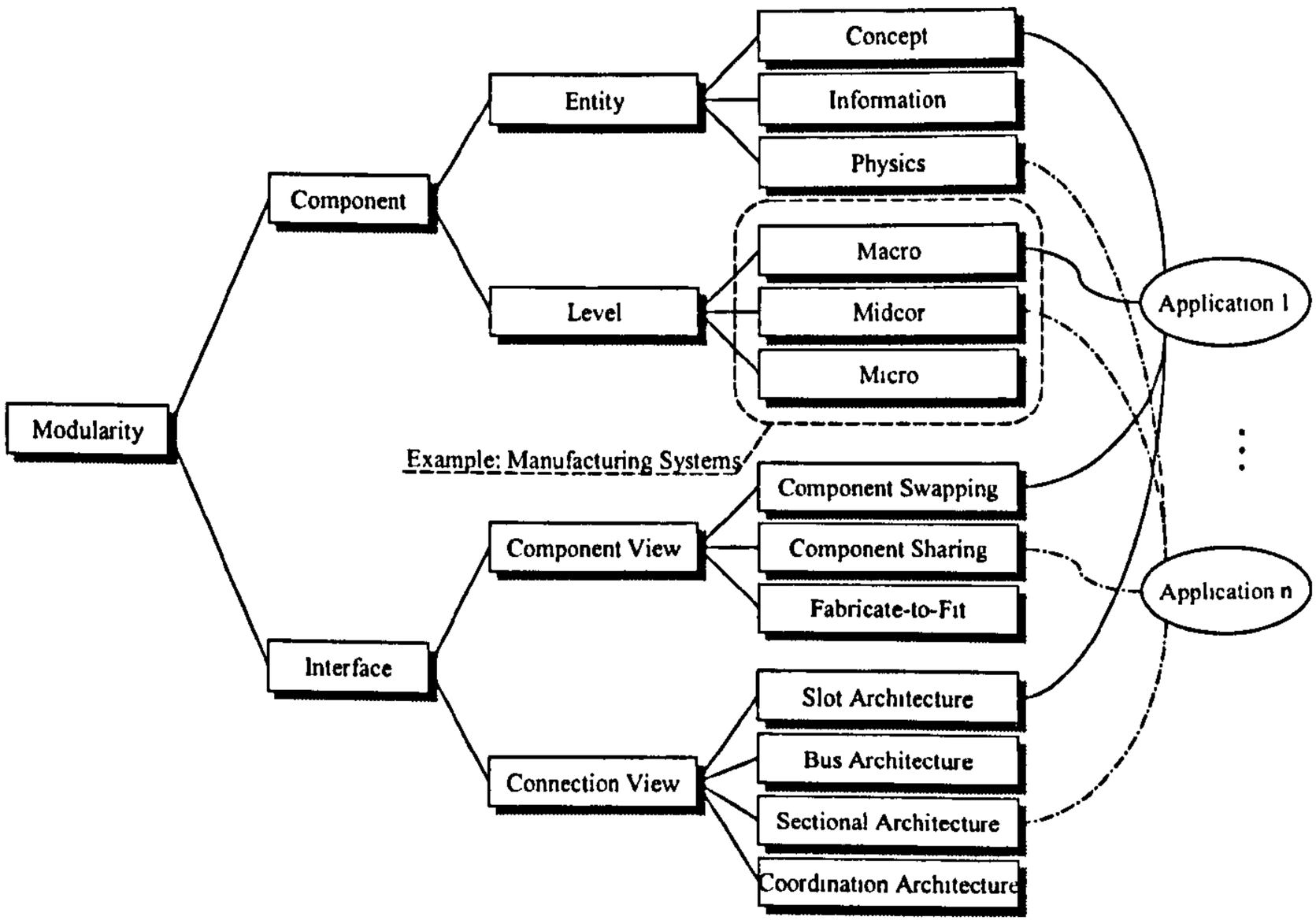


Figure 2.3 Taxonomy of modularity applications (Bi and Zhang [7])

2.2.1.3 Modular product/system development

Following the definition of modularity, two branches during the development of modular products and systems can be identified (Vos [133] and Bi and Zhang [7]). Figure 2.4 shows the two branches. Branch 1 is the definition and design of a modular system and the subsequent development of specific modules. This branch has to take the requirements for all possible systems into account. The second branch takes requirements for one specific product/system and combines the existing modules into a suitable system configuration.

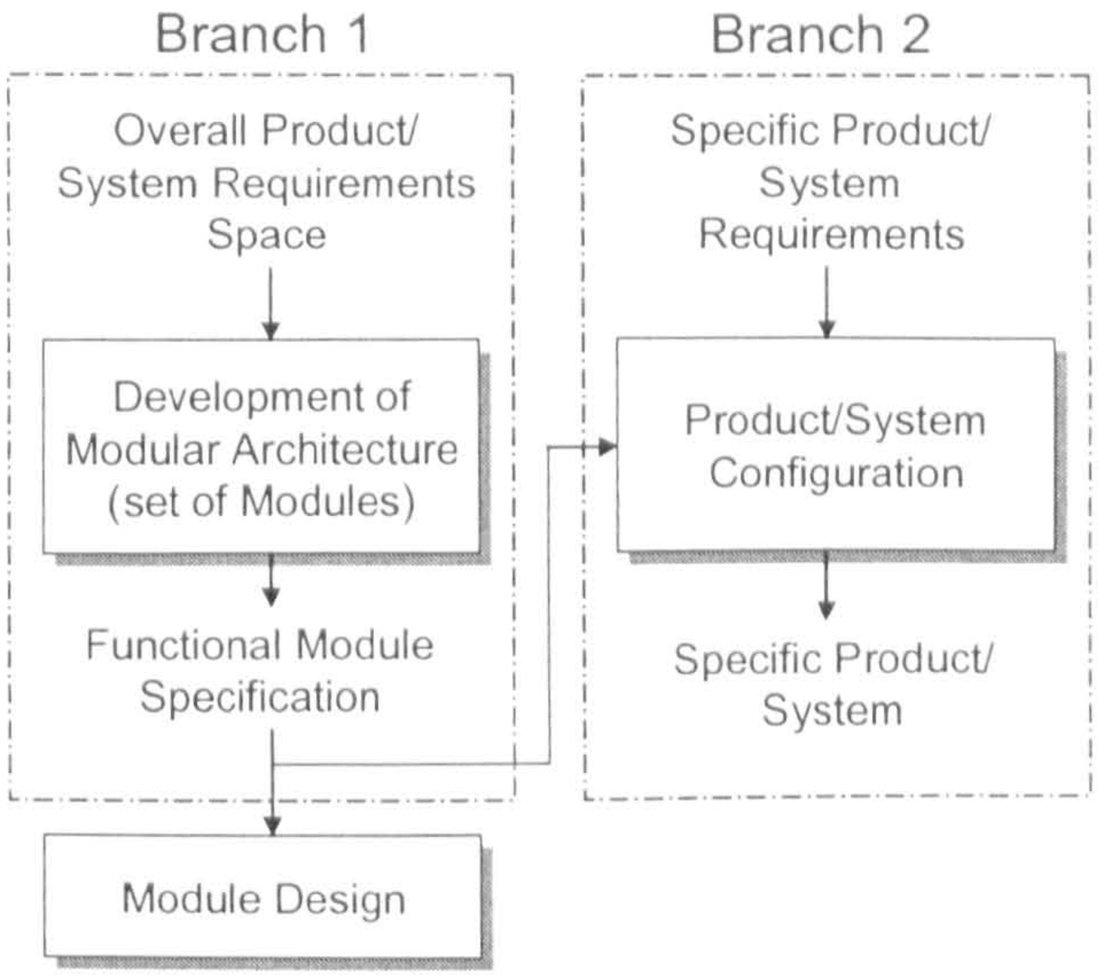


Figure 2.4 Modular Product/System Development (adapted from Vos [133])

2.2.2 Modular Assembly Systems

2.2.2.1 System Architectures

A significant research and development effort has being directed towards creating suitable system architectures for modular assembly systems (Boër, et al. [11], Giusti, et al. [43], Chen [18], Hollis and Quaid [53], Gaugel, et al. [40], Alsterman and Onori [3]). Modularity is one of the underlying technologies for reconfigurable and evolvable assembly systems.

Hollis and Quaid [53] define a modular assembly system structure based on cooperating 2-DOF robots. Their system consists of standardised autonomous workstations that build the basis components for the structure of their system. Actuator modules can be added to the system to provide the required process capabilities. They stress the need for a highly automated rapid configuration method as one of the basic requirements for successful reconfiguration of modular systems.

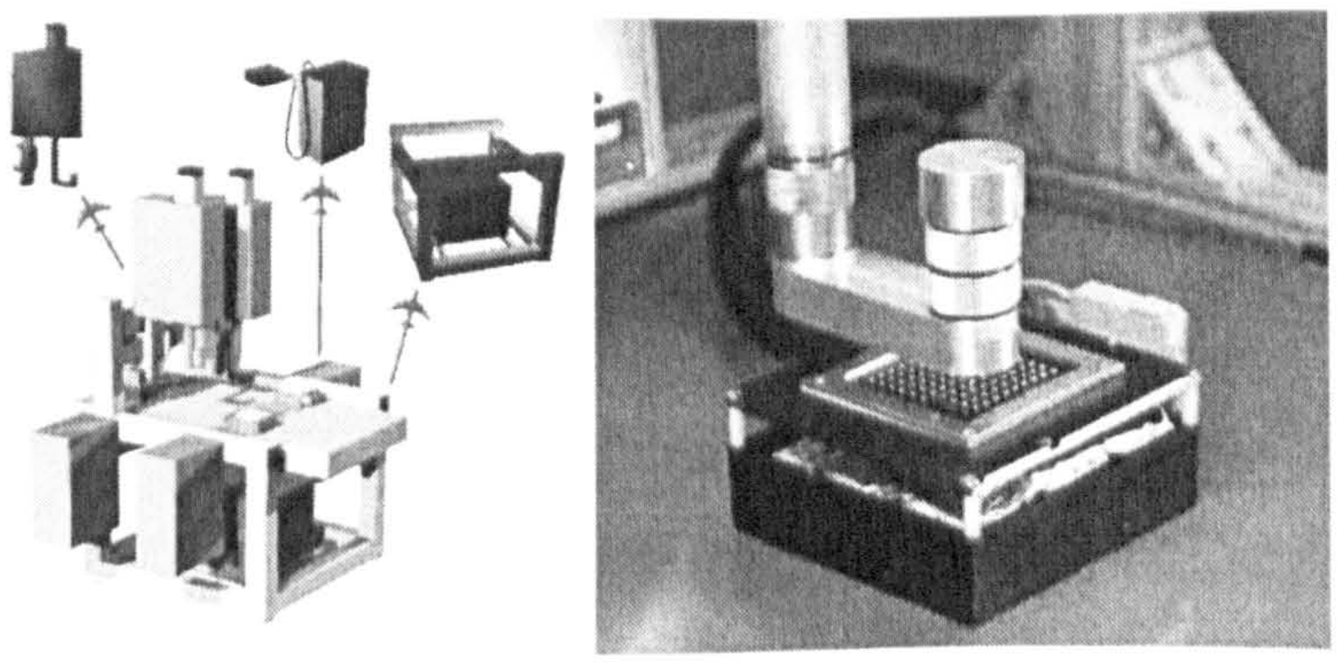


Figure 2.5 Agile Assembly Architecture (AAA/Minifactory [1])

Alsterman and Onori [3] introduce a platform for hyper-flexible assembly automation based on standardised components that can be configured into different assembly systems or workstations (Mark IV).

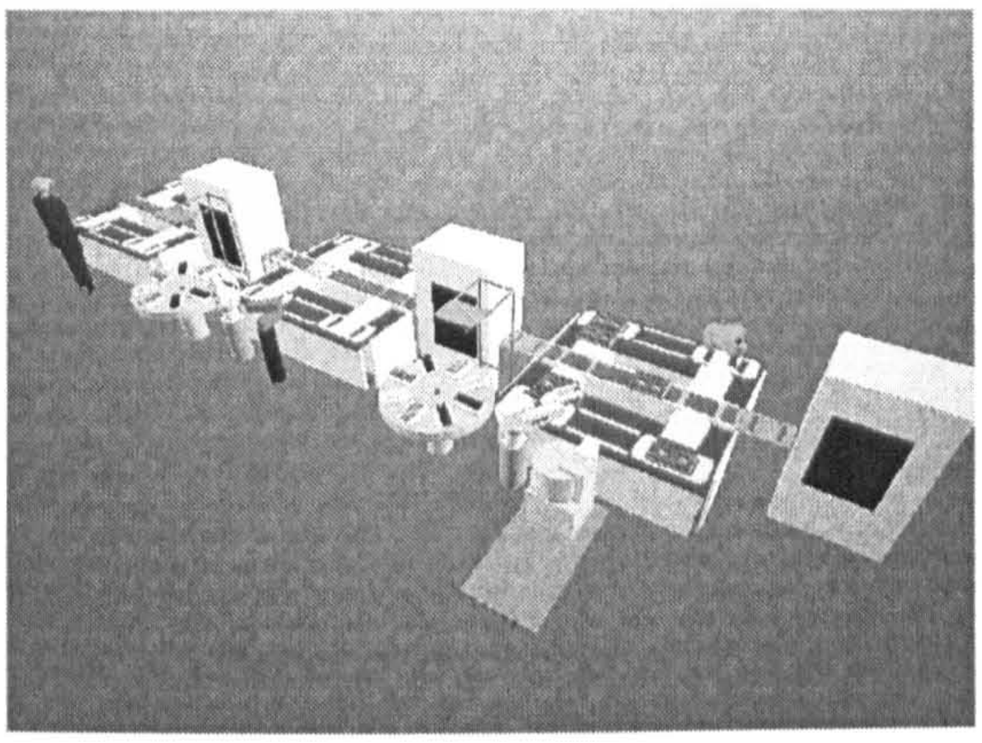


Figure 2.6 Mark IV Hyper Flexible Assembly System Architecture (Alsterman and Onori [3])

Giusti, et al. [43] report the development of a flexibly reconfigurable assembly cell. The cell is aimed to increase the accessibility of automated assembly cells for small and medium product volumes. The reported structure consists of the following main functional equipment entities: a 6 DoF robot, pneumatic screwing unit, hydraulic press, and 2 adjustable supports controlled along a vertical axis. The main feature of the proposed structure is its capability to self-reconfigure its component specific features by using its robot. Also the authors report the use of an integrated mechanical, electrical, and pneumatic interface between equipment modules and the table surface.

Gaugel, et al. [40] report a modular desktop assembly system using 2DoF planar motors for material transport and simple 2DoF manipulators for assembly and feeding (see Figure 2.7).

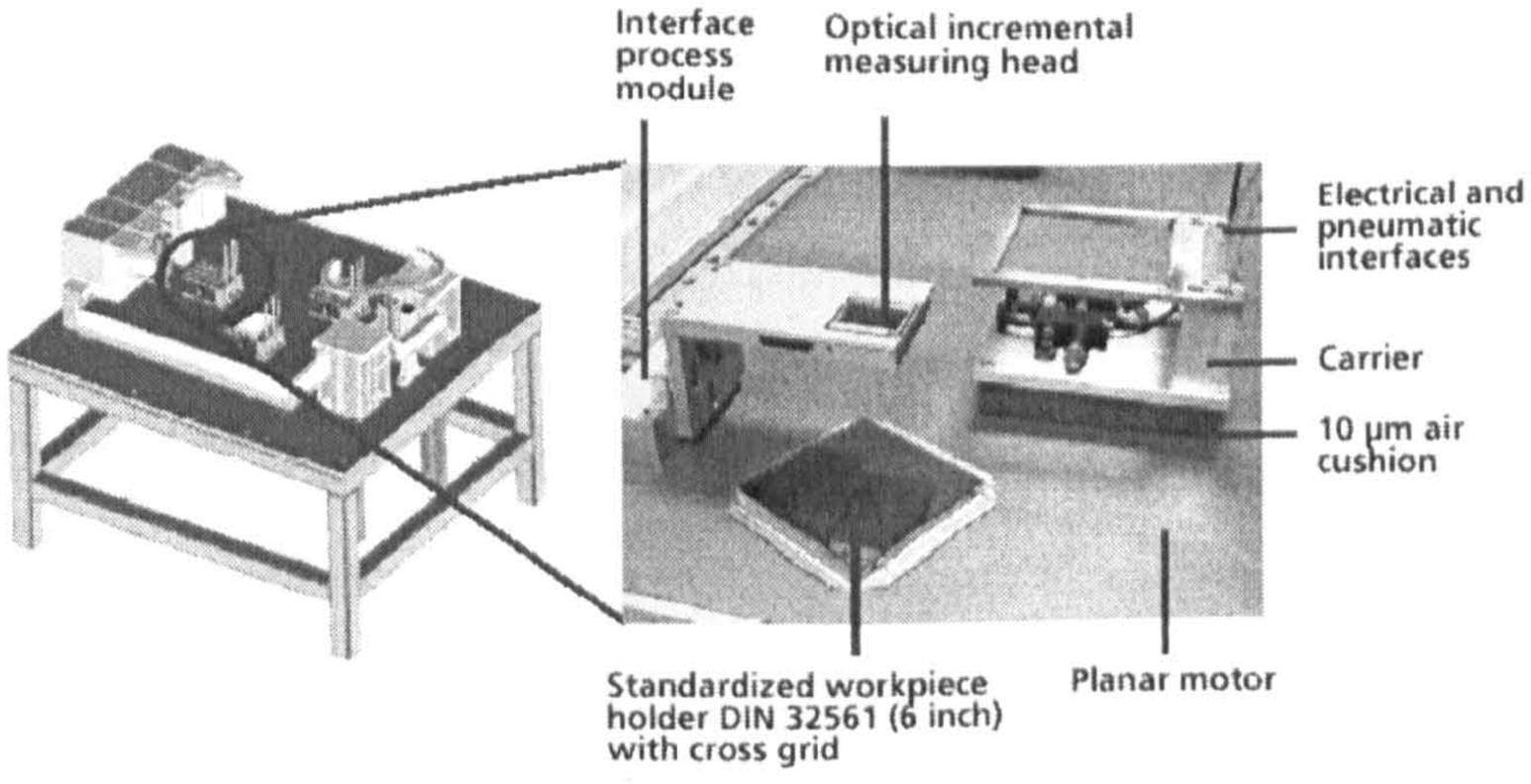


Figure 2.7 MiniProd Concept (Gaugel, et al. [40])

Barata de Oliveira [6] has reported a coalition-based control approach for agile re-engineering that looks at how a modular assembly system can be adapted to changes at shop floor level. Lastra [65] reports an agent based collaborative control approach that aims to facilitate the easier operation of modules on a control level (see Figure 2.8).

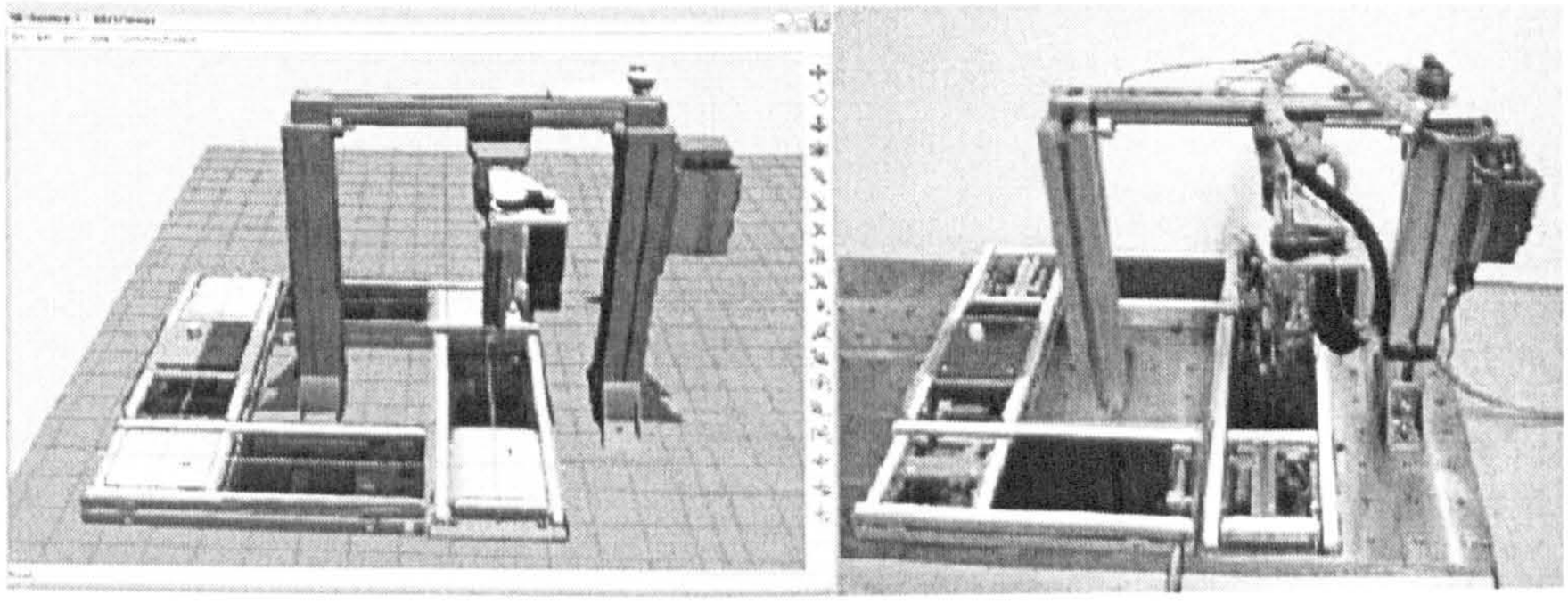


Figure 2.8 Actor-based Assembly System (ABAS) (Lastra [65])

Sugi, et al. [119] propose a holonic assembly system approach to reduce or even eliminate the configuration effort when new manipulators are added into an existing system (see Figure 2.9).

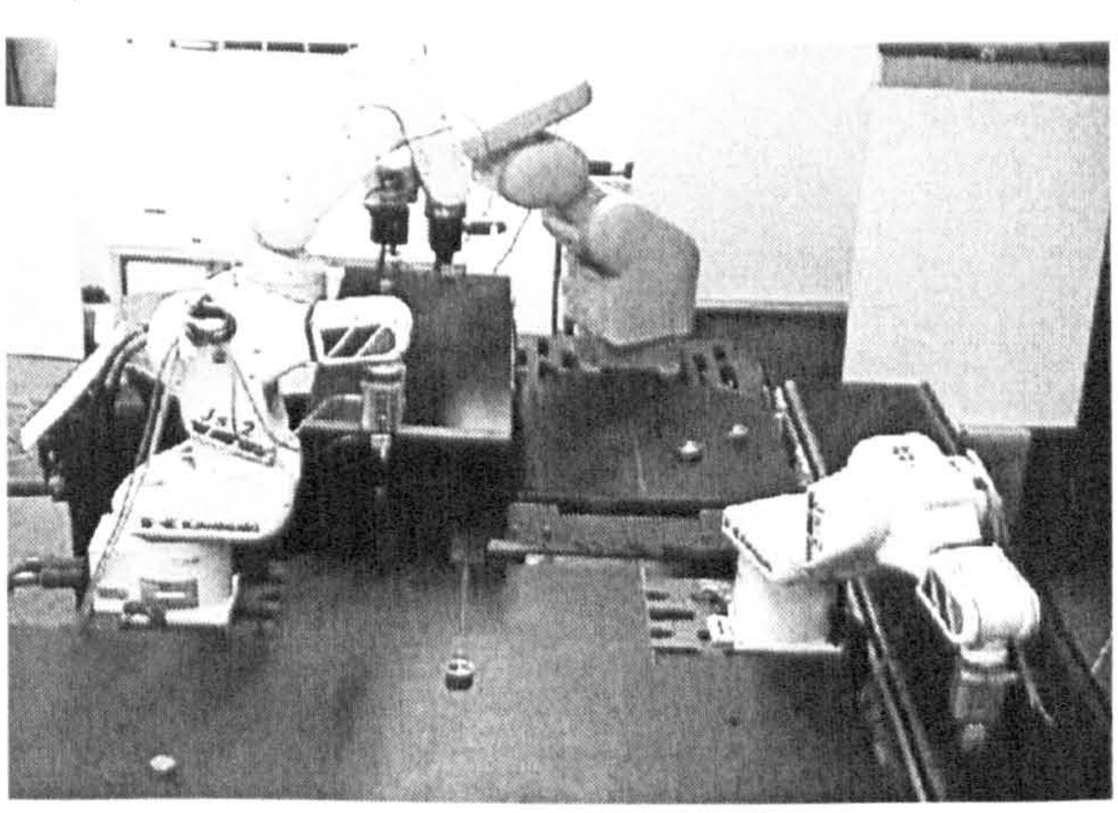


Figure 2.9 Holonic Assembly System Approach (Sugi, et al. [119])

The reported approaches aim towards addressing the specific control needs of EAS but do not explicitly consider the wider design decision-making environment.

2.2.2.2 Available System Solutions

Several examples of modular assembly systems are already commercially available as reported in the reviews of Onori, et al. [91], Alsterman [2], and Lastra [65]. They include: the TUFF system form ABB Flexible Automation (see Figure 2.10); the High Speed Assembly Cell (HiSAC) form Cencorp (see Figure 2.11); the Sony Smart Cell

(see Figure 2.12); the Flexline from SMH Automation (see Figure 2.13); the Mikron-Syfast assembly system (see Figure 2.14); and others.

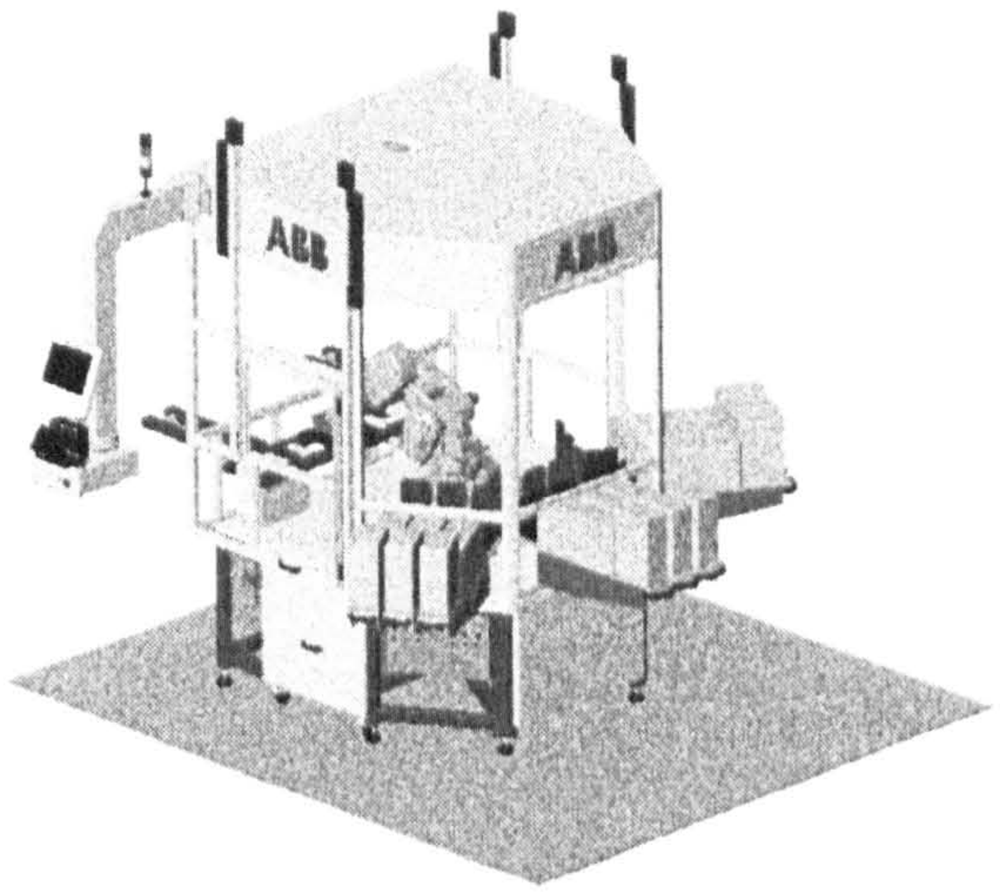


Figure 2.10 ABB TUFF system

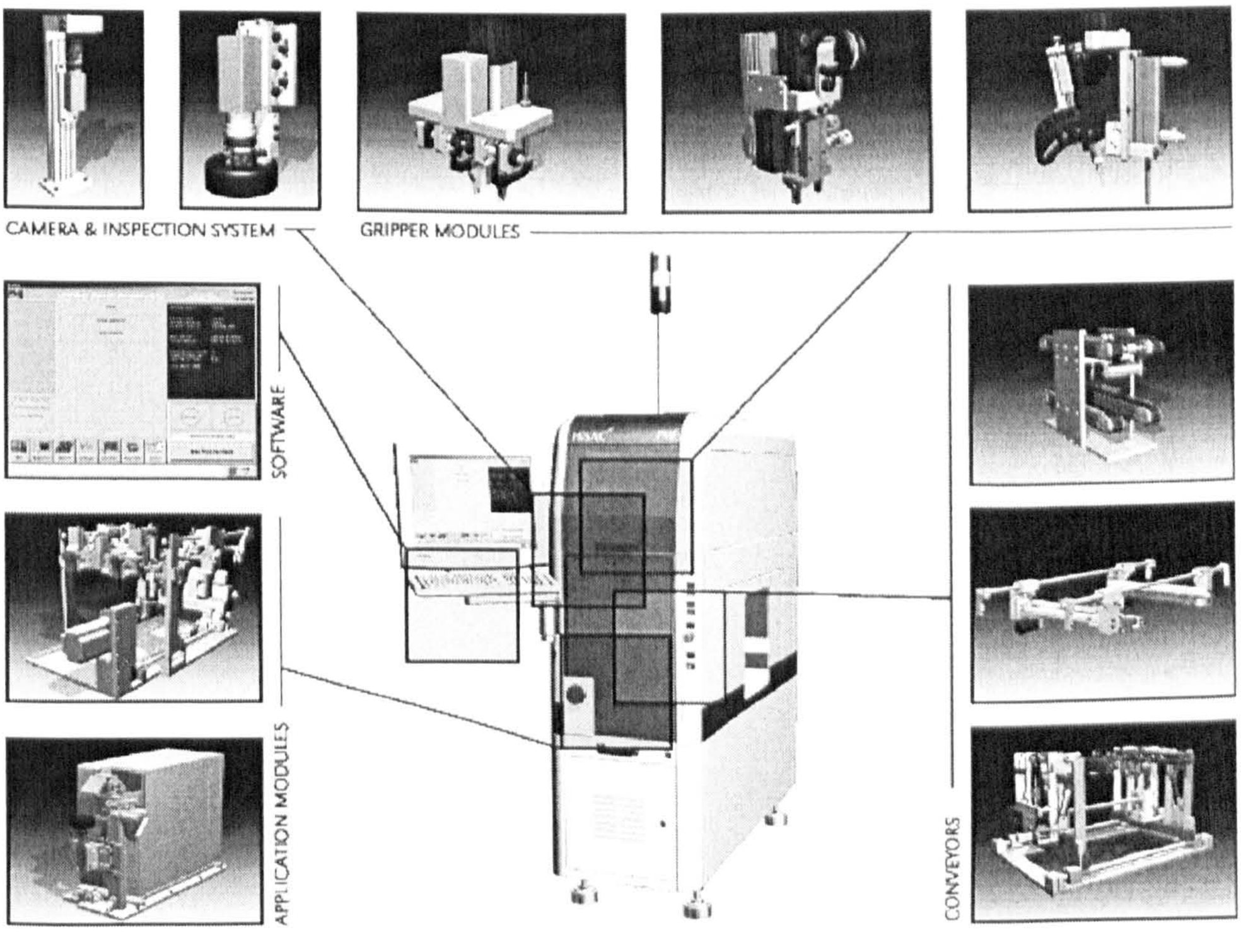


Figure 2.11 HiSAC-500 High Speed Assembly Cell (Cencorp [14])

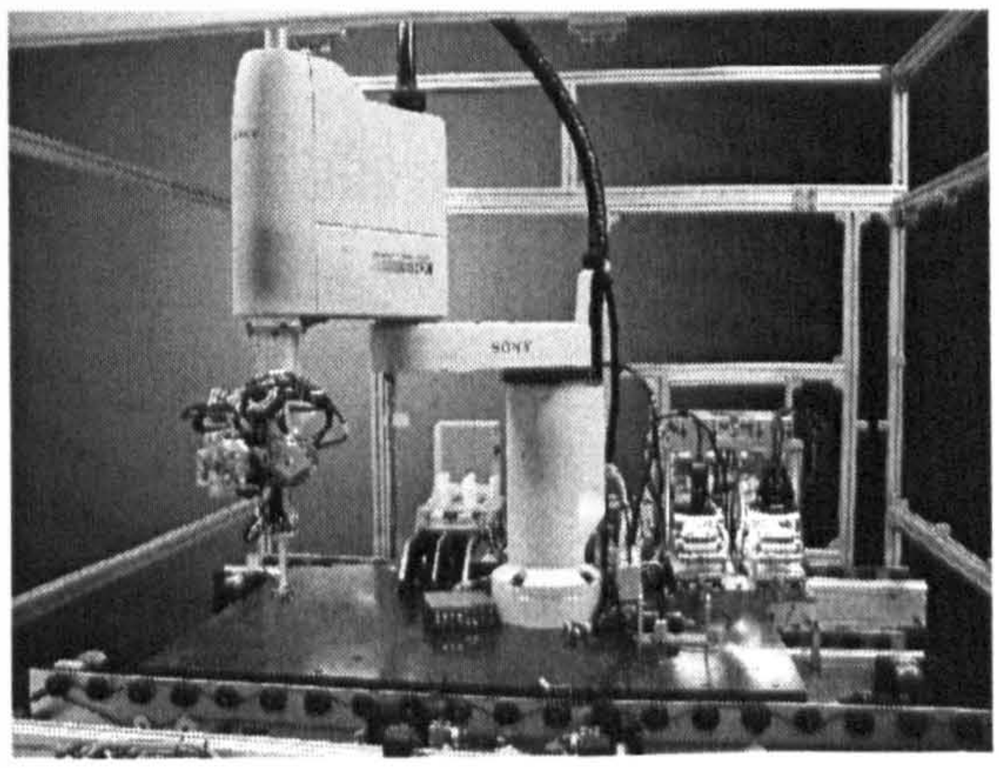


Figure 2.12 Sony SMART Cell (Fujimori [39])

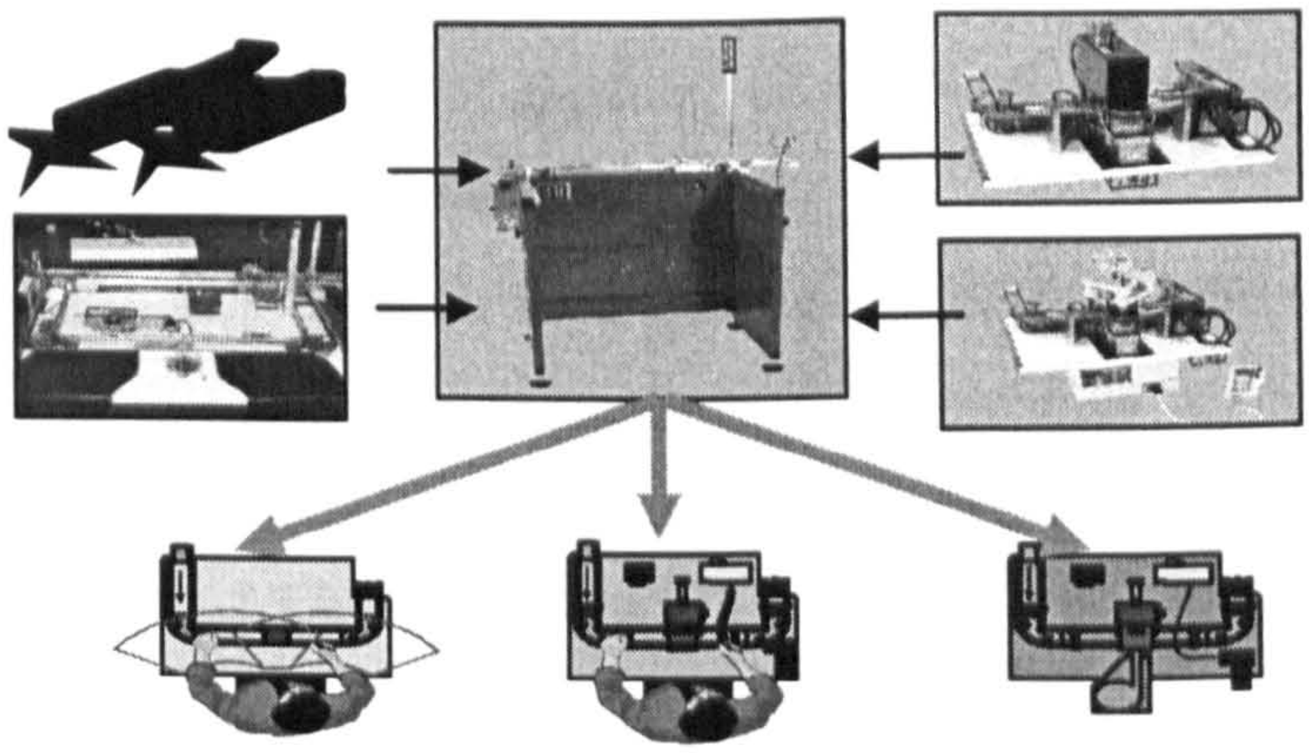


Figure 2.13 SMH Plug&Produce™ concept

Figure 2.14 Mikron-Syfast assembly system (Frauenfelder [37])

The reported system architectures and commercially available systems fulfil the basic structural requirements for EAS but still fall short on the control and design side. This shows the current trend towards a reconfiguration and evolvability based school of thought but also highlights the necessity for further research into enabling technologies especially on the design side.

2.2.2.3 Modular Equipment

Research in the area of assembly equipment focuses on two main areas: the development of new equipment solutions and the configuration/design of task specific equipment from well defined elements. This review focuses primarily on the latter.

The most dominant domain within equipment design area is the task-based configuration of automated manipulators, so called robots. Other areas of research are the configuration of fixtures, feeders and tools (O'Shea, et al. [92], Shirinzabeh [116], and Joneja and Lee [60]). Across these areas the research effort is focused on methods and tools that can help define, optimise and control equipment solutions for specific assembly problems/tasks. A set of modular or standardised components generally builds the basis for the proposed methods or tools.

In the area of robotics research there are two main disciplines: industrial robot configurations and self-reconfiguring robots. The research in the area of self-configuring robots is mainly concentrating on the development of autonomous low DoF modules that have active interfaces which are controlled by the modules (Ünsal, et al. [130], Chiang and Chirikjian [20], Murata, et al. [85]). Through the use of their degrees of freedom in combination with their interfaces the modules can re-arrange themselves according to instruction from a supervisory control. The technology of self-reconfiguring robots, however, is not yet mature enough to be applicable for use in industrial assembly.

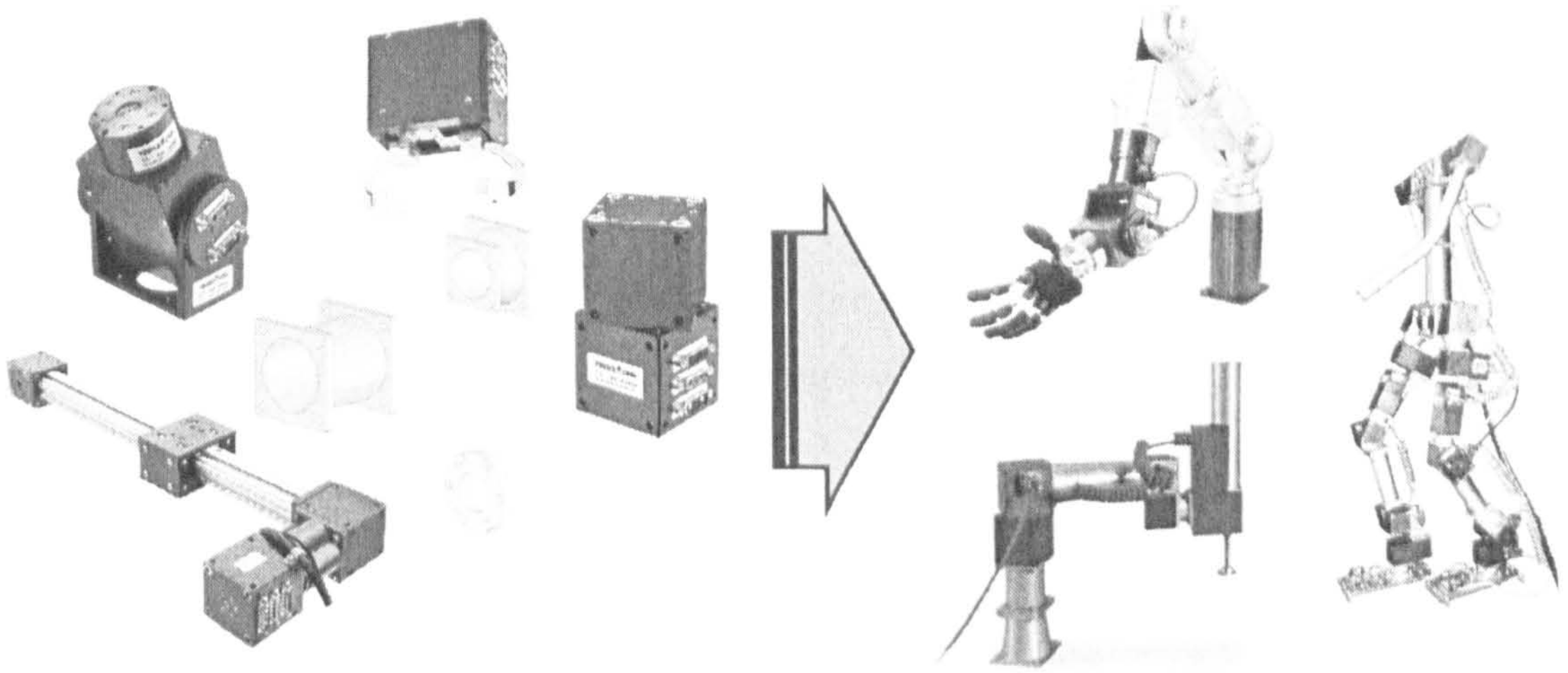


Figure 2.15 Application Examples of Modular Robot Structures (Amtec [4])

Research in the area of industrial robot configuration has focused on the development of standardised robotic modules that can be configured to generate different configurations depending on the required application (Chen [18], Bi and

Zhang [7], Yang and Chen [139], and Paredis, et al. [95]). The three main problems which have been looked at are: feasibility checking, optimisation and control generation for robotic configurations. Starting point for the configuration is generally a specific task, which is normally expressed as a set of working points for the robot including parameters like posture, work load, accuracy etc. at each point.

Different modular robot solutions are available today. The most noteworthy are probably the PowerCube™ developed by Amtec robotics (see Figure 2.15 for an example). They have found several applications especially in the academic domain.

The focus of the reported work in the area of modular assembly systems has mainly been on the physical structure and control aspects that enable reconfiguration. They highlight modular equipment solutions as one of the fundamental requirements for reconfiguration and hence evolvability. The design and decision making aspects necessary for requirements driven rapid reconfiguration were mostly outside the scope of the reported work and clearly need further investigation.

2.2.3 Design Methodologies

Suh [120] has presented an extensive definition of an axiomatic approach to engineering design based on his experience in manufacturing. He distinguishes between axiomatic and algorithmic design methods as discussed by Jacquet, et al. [59]. The axiomatic design approach defines domain models and rules which are used to generate a new design.

The algorithmic design approach as proposed by Pahl and Beitz [94] is defining a set of steps that need to be completed during the design process. Problem decomposition and synthesis approaches or hierarchical design approaches are aimed to deal with the inherent complexity in large system engineering (VDI 2221 [131], Pahl and Beitz [94], Stevens, et al. [118], Roth [110], Tomiyama, et al. [123]). An overview of the principle approaches is shown in Figure 2.16 and Figure 2.17. The fundamental approach is to decompose the overall design problem into sub-problems until a satisfactory level of simplicity and clarity has been reached that allows the definition of sub-solution alternatives. The sub-solutions are then being combined and synthesised to create higher level solution alternatives. The synthesised solutions are validated against the problem definition on the same level of complexity. If the solutions are found to satisfy the original problem definitions they are again combined

until the full system has been defined. If the lower level solutions do not match the requirements, they need to be refined until a match has been achieved.

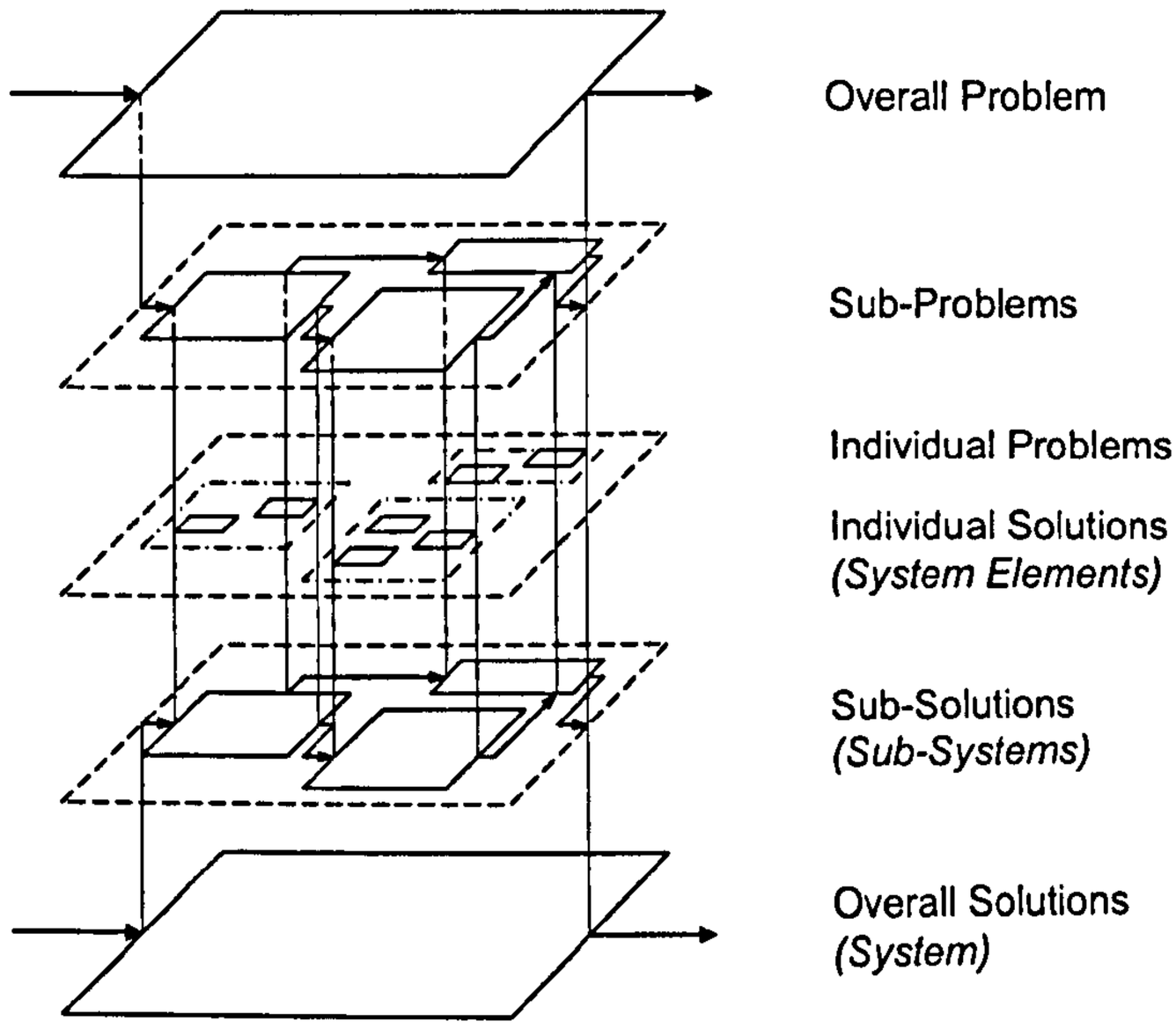


Figure 2.16 Problem decomposition and solution synthesis (VDI 2221 [131])

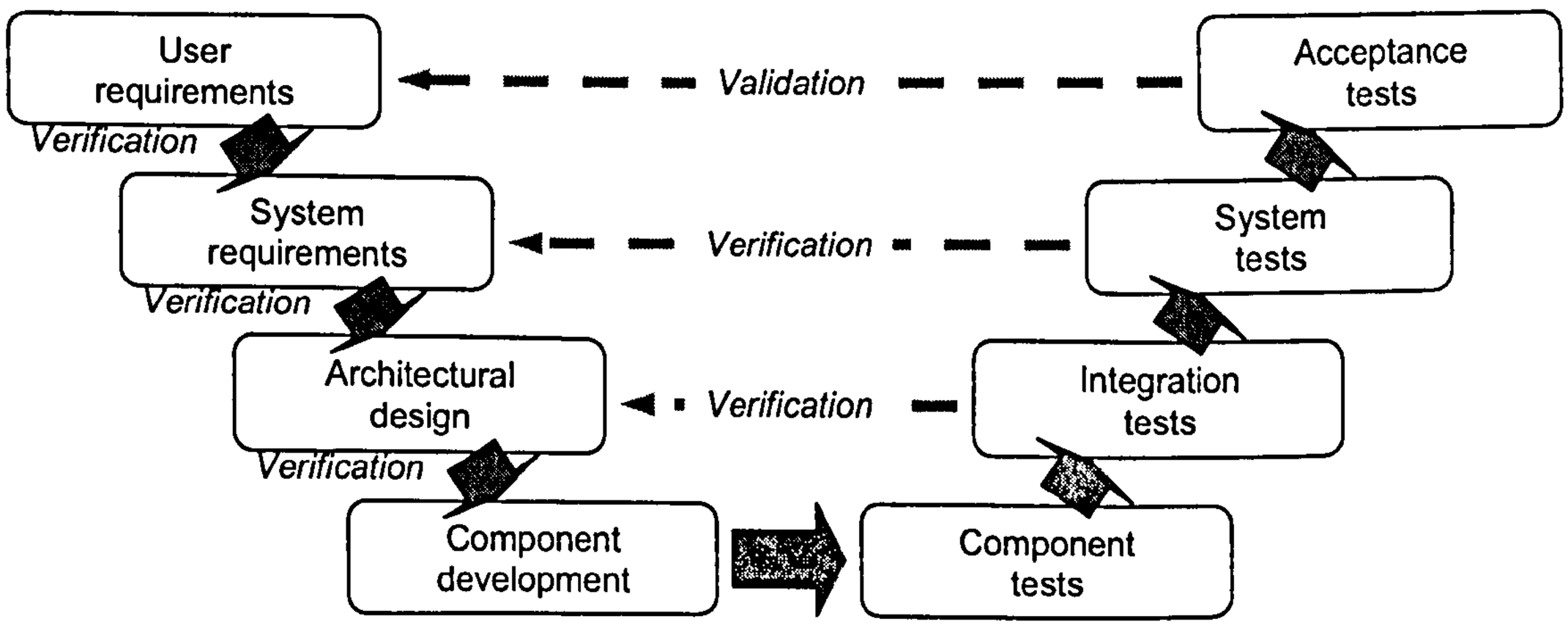


Figure 2.17 The V-Diagram for a simple design life-cycle (Stevens, et al. [118])

Jacquet, et al. [59] recognise that the axiomatic method explains the design context but not how to proceed, whereas the strictly algorithmic methods tend to restrict the concurrency of the design process.

Stevens, et al. [118] illustrate the system engineering approach to design of complex products and system. This is a holistic approach looking at the design as part of a products/systems life cycle. The approach is more commonly used for complex products/systems but is not restricted to them.

Rosenman and Gero [107] discuss the socio-cultural and techno-physical aspects of the design using purpose-function-behaviour-structure as basis for the argumentation. The paper explains the definition of purpose using the distinction between the socio-cultural and techno-physical environment (see Figure 2.18).

In the paper purpose, function, behaviour, and structure aspect of the model are defined with the relationships between them. "STRUCTURE exhibits BEHAVIOUR effects FUNCTION enables PURPOSE. PURPOSE enabled by FUNCTION achieved by BEHAVIOUR exhibited by STRUCTURE."

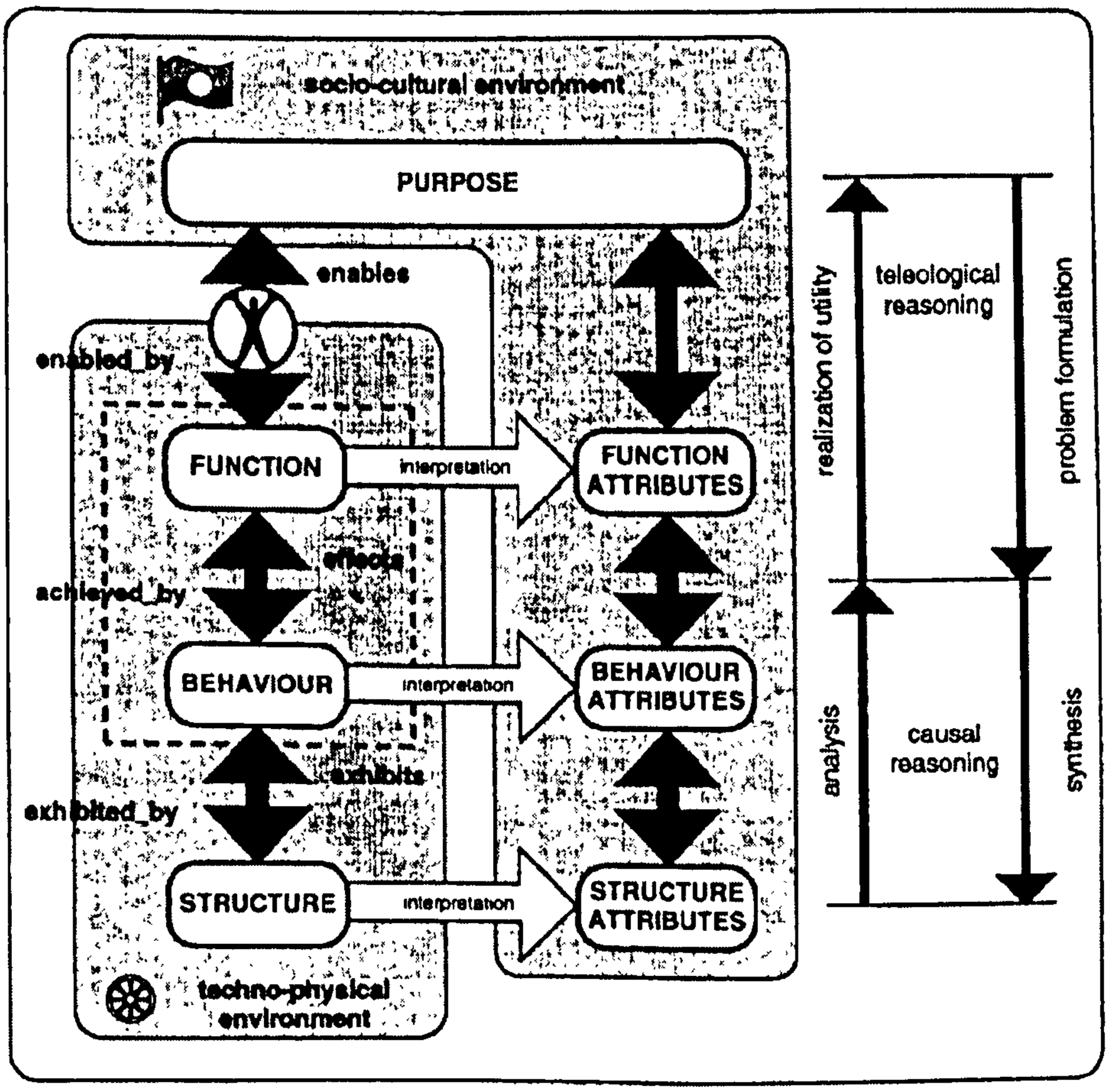


Figure 2.18 The concepts, environments, and processes in design (Rosenman and Gero [107])

The authors also discuss the application of the purpose-function-behaviour-structure model in the design process. They outline the transition from the purpose of an artefact to its eventual structure. The design process is described as iterative formulation-synthesis-analysis-evaluation cycles. They discuss the ill-structured nature of design, classification of design objects, decomposition and formulation, and specialisation in more detail.

The functional definition of an artefact builds the foundation for the integration between different design domains (aspects associated to groups of experts, as for

example architects, civil engineers, and contractors) within their model. The role of intended, non-intended, and emergent functions is being discussed. The application of the proposed modelling philosophy is also being analysed for multidisciplinary design.

Specifically for the design of assembly systems the decomposition process involves a simultaneous detailing of the assembly process and a grouping of necessary process steps into equipment requirements. The selection of appropriate equipment solutions is in the first instance based on their capability to achieve the required set of assembly process steps. Only in the second instance is the specific behaviour of the different solution alternative taken into consideration to optimise the performance of the overall system. The synthesis and validation is again mainly driven by the combined process capabilities of the chosen equipment solutions.

In the design of modular systems, the decomposition process can be guided towards possible solutions within the chosen system architecture. In a modular system only solutions that can be achieved through combination of existing modules within the given architectural framework are possible. The possible solution space is determined before a specific design problem arises. The knowledge about the resulting constraints can be used during the decomposition of the specific design problems. This has the added advantage that problems which cannot be solved with the chosen modular system architecture can be spotted already during the requirements decomposition.

Reconfiguration is a natural extension of the modular system design with the primary difference that it needs to take the existing system into consideration (Koren, et al. [64]). Reconfiguration approaches need to reduce the effort of change between the old system configuration and the new set of required capabilities. This means that the responsibilities of the equipment entities within the overall assembly process of an assembly system need to be clearly defined.

Generally it can be seen from the different design methodologies that they require for basic mechanisms: specialisation, decomposition, configuration, and synthesis. In the following sections those aspects will be addressed in more detail before looking at some existing design frameworks and tool sets.

2.2.4 Ontologies

An ontology is concerned with the study of being or existence and their basic categories and relationships, to determine what entities and what types of entities exist. It therefore has strong implications for conceptions of reality (Wikipedia [136]). They help to clarify a domain's knowledge structure and therefore improves knowledge sharing, utilisation of captured knowledge, and maintenance of existing knowledge (Chandrasekaran, et al. [15], Grunninger and Lee [48]). Those aspects are very important for the definition of knowledge modelling frameworks for the support of distributed engineering design decision making environments. Many ontologies have for this reason been developed in the engineering domain in recent years (Borst, et al. [12], Mizoguchi and Kitamura [84], Ciocoiu, et al. [21]).

A number of different ontology definition languages have been proposed. They include: Knowledge Interchange Format – KIF (Genesereth and Fikes [42]), Ontolingua (Farquhar, et al. [34]), CommonKADS CML (Schreiber, et al. [114]), CycL (Lenat and Guha [66]), DAML + OIL (DAML [24]), and OWL (OWL [93]) to name the most important ones. The most recently developed and also the most promising ontology language is OWL. This ontology language is aiming to enhance the knowledge content of web resources. It has been developed by the W3C consortium and is utilising on the widely accepted representation languages XML and RDF.

The CommonKADS CML graphical notations will be used to define the proposed new domain conceptualisations. The advantage of the CommonKADS representation is that it goes beyond the definition of facts and rules and also includes graphical notations for inference and reasoning activities. The basic description of concepts in the CommonKADS framework is based on UML notations which makes it more readable for a wider audience.

The CommonKADS knowledge modelling language and graphical notations (CML) cover most of the required aspects for this work but do not address some of the modelling requirements for distributed decision making. The language and graphical notations have been extended to overcome this shortcoming where required.

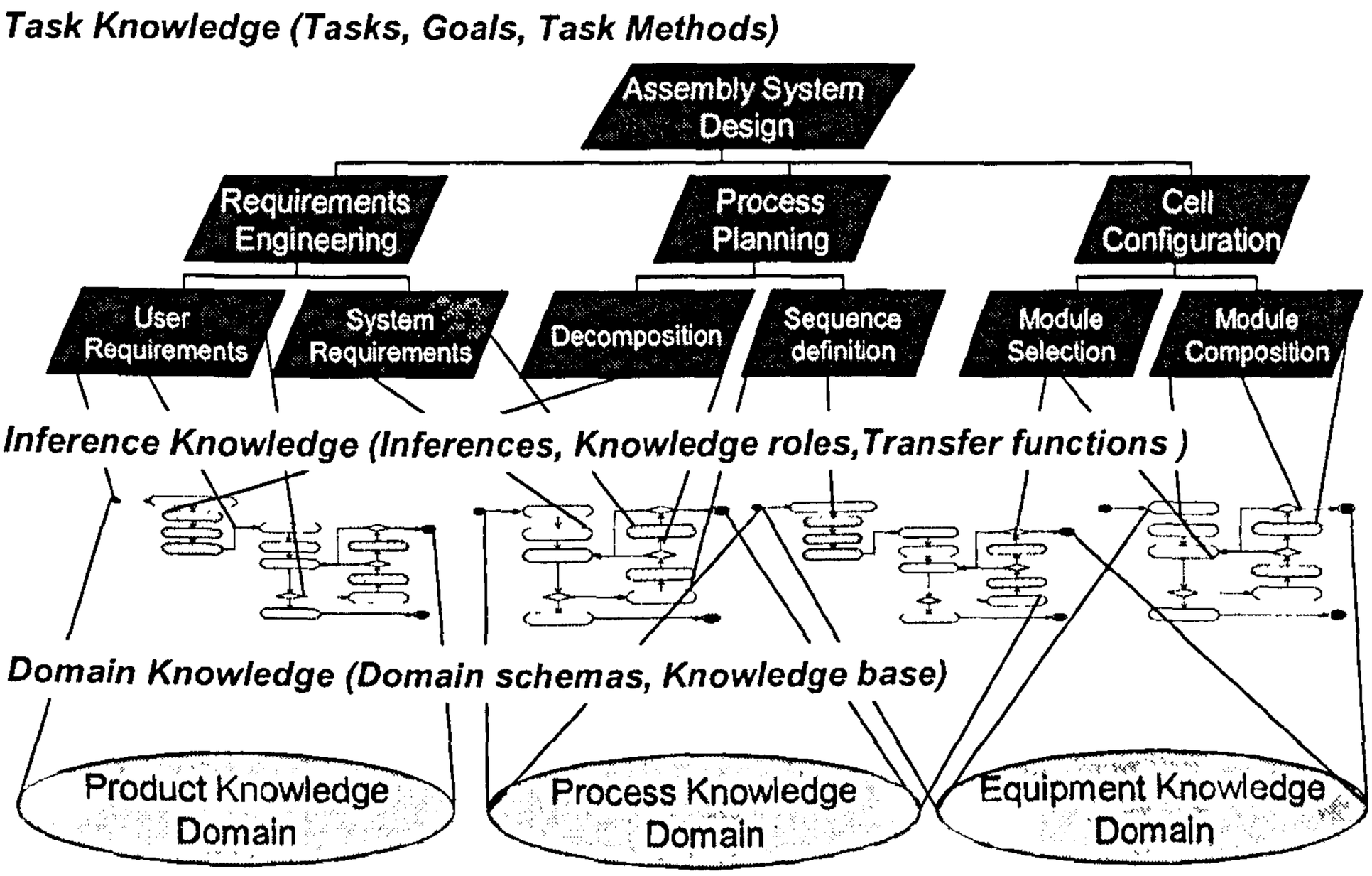


Figure 2.19 CommonKADS model of the assembly system design process (adapted from Ratchev [104])

Following the CommonKADS knowledge structuring approach the different aspects of the proposed assembly system design methodology have been modelled on three knowledge definition levels (see Figure 2.19):

- **Task knowledge level,** defining the design tasks required for achieving specific design goals and who is involved in them (actors). Task knowledge is described on different levels of abstraction defining a hierarchy from general tasks to more and more specific tasks. At the lowest level are all those tasks that cannot be further decomposed into subtasks. They are entirely built from members of the inference knowledge level. Each task has one or more task methods that define their sub-activities and in which order they need to be performed. Actors are mapped to all the tasks they are involved in.
- **Inference knowledge level,** defining the inferences, decisions, and communication acts required for performing the design tasks on the task knowledge level. Inferences are defined through the type of their dynamic input and output knowledge and a set of rules that are used to infer the output from the input. Decisions are specialised inferences that have dynamic input knowledge and infer a yes-or-no decision based on a set of static rules, the decision criteria. Communication acts define what type of knowledge can be exchanged between reasoning activities of different actors.

- **Domain knowledge level,** defining all the concepts, relationships, attributes and rules which are used by the inference level activities. Concepts are defined as a set of attributes. Relationships are either concepts in their own right or are defined through the attributes of other non-relationship concepts. Rules are defined as through their antecedent, consequent and the causality between them. The constraints of the model are specified as first order logic axioms.

A number of ontology development frameworks have been reported: Ontolingua Server (Farquhar, et al. [34]), eKADS (eKADS [31]), and Protégé (Protégé [97]). They assist with the definition and maintenance of ontologies. eKADS allows the definition of CommonKADS knowledge models. It does provide, however, only limited support for the utilisation and verification of the defined models. Protégé is an ontology definition and instantiation framework that provides a software environment for the definition of ontologies. The ontology can be instantiated and made available as a knowledge base for both internal and external utilisation. Protégé allows the definition of both frame-based ontologies as well as OWL-based ontologies.

Protégé has been chosen to implement and test the new ONTOMAS formalisms proposed in this thesis. The choice was based on the capability of the Protégé framework to include plug-ins for axiom checking and inference reasoning. Furthermore, the frame-based ontology representation of Protégé was chosen to keep in line with the CommonKADS notations used throughout this work.

2.3 State of the art

After having discussed the general background of this thesis, the current state-of-the-art in the specific areas treated by this thesis is being examined. The focus of this thesis is on the development of suitable domain ontologies for assembly process and modular equipment specification which are suitable for the integrated design of modular assembly workstations. The following sub-sections give a brief overview of the reported work in assembly process modelling, equipment modelling, and integrated design methods.

2.3.1 Assembly Process Specification

The assembly process specification deals with describing the characteristics of individual process steps and defining their temporal order. Process specification

languages need to provide formalisms for the definition of the topological process structure to describe their sequential order, the hierarchical structure to introduce different levels of detail, and the classification of different process types to enable a domain specific interpretation of process models.

A number of different general process specification languages have been reported. They include: the Sharable Plan and Activity Representation - SPAR (Tate [122]), the high-level robot programming language GOLOG (Levesque, et al. [67]), the Workflow Process Definition Language – WPDL (WPDL [137]), and the Process Specification Language – PSL (Schlenoff, et al. [113], Grüninger [47]). All these proposed languages are very general and do not take the specific aspects of the assembly domain into consideration. The most prominent of those languages is PSL which has been developed by NIST to facilitate the exchange of process based information between manufacturing systems. Some of the basic notations and axioms of PSL have been applied in this work.

The definition and planning of assembly sequences is a problem that has received a lot of attention in the assembly domain. It studies how the order in which the components of a product could be assembled can be determined and optimised. The activity of putting two components together is commonly referred to as assembly task. A number of different approaches to represent assembly task sequences have been used. The most often used assembly sequence representations are precedence relationships, directed graphs, and And-Or-graphs. They are related to each other and can be transformed into each other (Homem de Mello and Sanderson [55]).

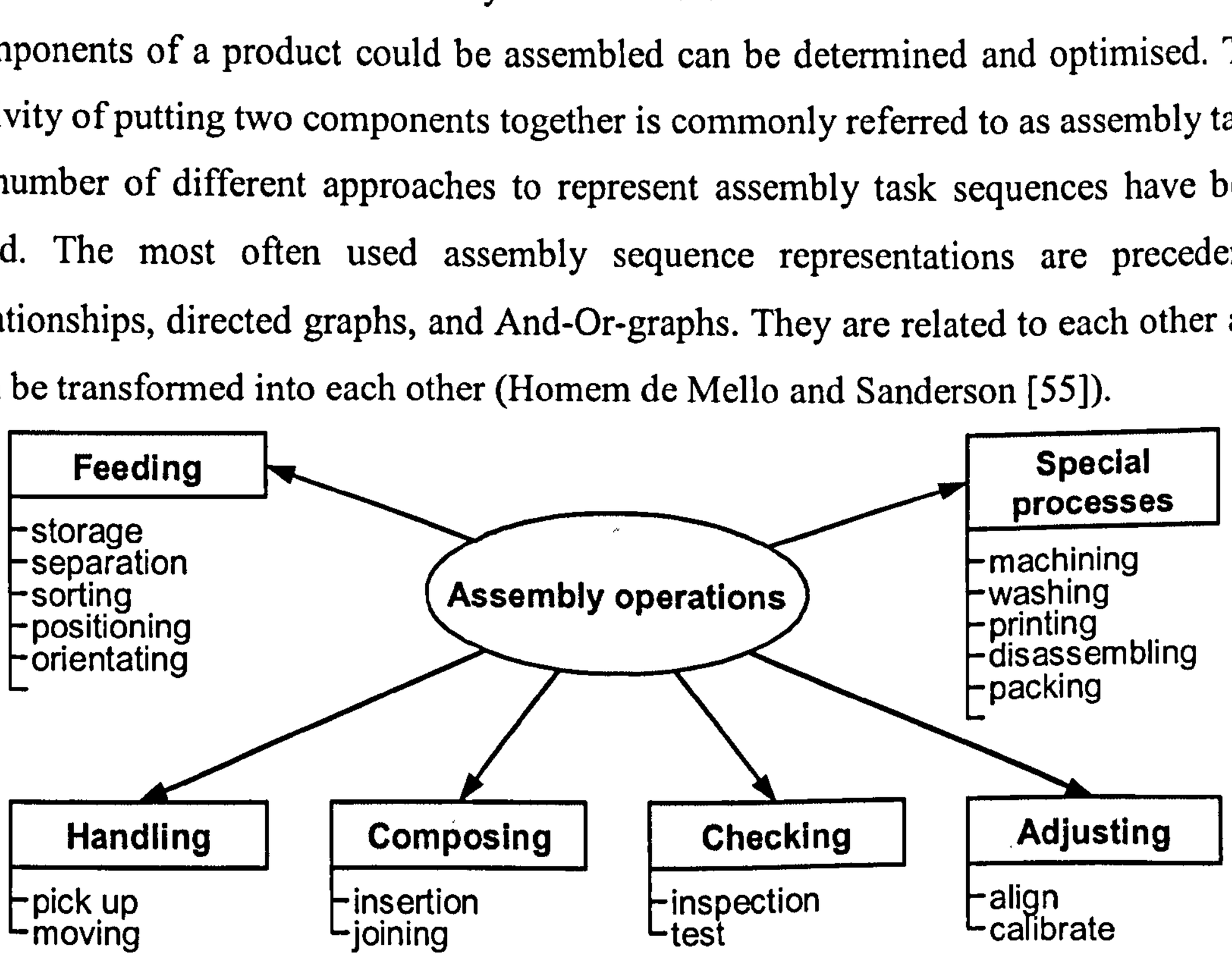


Figure 2.20 Assembly Operations Overview (Rampersad [99])

The assembly sequences do not address the individual steps that need to be carried out to achieve the assembly of two components. Rampersad [99] proposed to split

each assembly task into a number of assembly operations that reflect the basic activities which need to be carried out in an assembly system to assemble two components. He classified the assembly operations under six main types as shown in Figure 2.20. Vos [133] also stresses the need to further decompose assembly tasks to capture the requirements for equipment selection. He defines an operation classification that does not clearly distinguish between taxonomical and hierarchical formalisms. Lohse, et al. [70] suggest structuring the assembly process on three distinct hierarchical levels (tasks, operations, and actions) that can be linked to corresponding levels on the equipment side. Barata de Oliveira [6] uses a set of basic skills which are clustered into more complex, higher level skills to describe an assembly process.

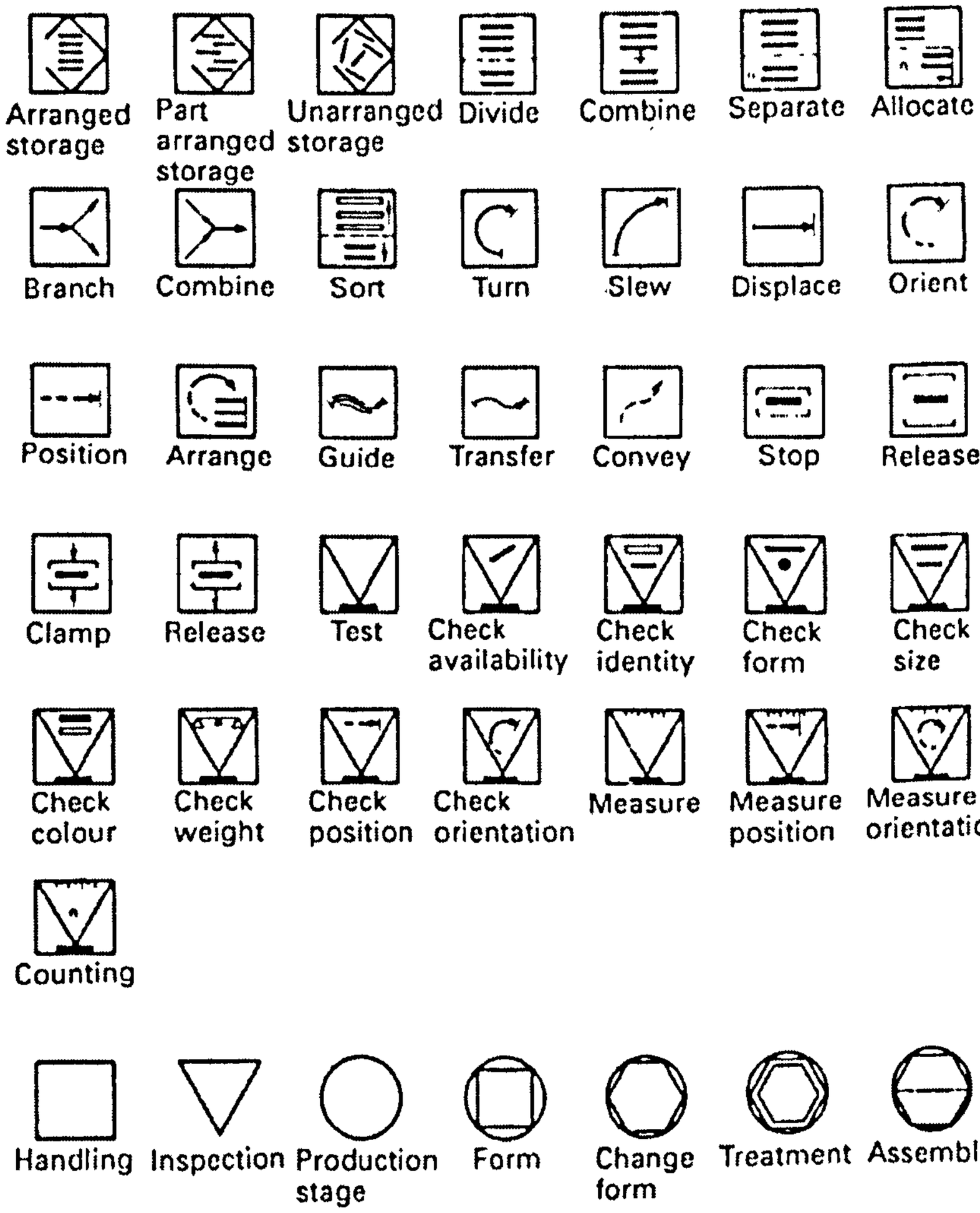


Figure 2.21 Part function symbols for handling VDI 2860 (Lotter [77])

The joining or assembly operation is the central aspect of the assembly task definition. A comprehensive classification of different types of assembly operations is given in DIN 8593-0 [28]. A number of basic handling and logistic operations have

been classified in VDI 2860 [132]. Figure 2.21 shows an overview of the translated basic activities defined by the VDI 2860 as given in Lotter [77].

Despite the significant work in the area of process modelling there is still a need for more comprehensive assembly process domain conceptualisations. The decomposition and classification of assembly processes at different levels of hierarchy need to be further explored to allow the process model to become the backbone of modular assembly system specification and selection as has been suggested by (Vos [133], Onori, et al. [91], Barata de Oliveira [6], Lastra [65]).

2.3.2 Equipment Modelling

There is a general consensus that one of the critical enabling factors for highly automated design systems is the availability of equipment models that provide the required information for equipment selection and system integration. Several integrated equipment models have been reported which focus on different aspects of modular assembly system design using an object-oriented paradigm (Rosenman and Wang [109], Yoshioka, et al. [140], Zhang and van der Werff [145], Lohse, et al. [71]).

Rosenman and Gero [108] propose a multiple functional view of artefacts to facilitate a collaborative design process. The approach is centred around a Purpose-Function-Behaviour-Structure model. Different functional representations for different domain experts are proposed to express their view of the artefacts to be designed. The relationships between purpose, function, behaviour, and structure are discussed and defined. The design process is described as an iterative transition from the purpose space over the functional and behavioural space to the structural space.

Qi, et al. [98] propose a General Engineering Data Model (GED) to facilitate the sharing and exchange of engineering data during the design of special purpose machines. The engineering data from different engineering disciplines is associated to representations of the mechatronic components in the modelled machines. This enables an easy extension of the model to include new discipline-specific aspects of the component without compromising the existing definitions.

A number of domain specific equipment models have also been reported focusing at different aspects and or levels of abstraction of the equipment characteristics. Zhang, et al. [144] have developed a model that focuses on the representation of

robots and their working environment. They defined an object-oriented representation of modular robots expressing the capabilities of its modules in a function-behaviour-structure model. The required capabilities are defined as a set of requirements, which have not been defined.

Zhang and van der Werff [145] and Neville and Joskowicz [88] report models for the specification of mechanisms focused on their mechanical/kinematic aspect. Neville and Joskowicz [88] developed a representation language for fixed axis mechanism specification. The language addresses the definition of structure/geometrical and behaviour aspects of the mechanisms. The behaviour definition provides concepts for kinematic and simple dynamic descriptions based on motion primitives and aggregates (parallel and sequential). The language is used for automatic design validation. The validation compares the desired behaviour (output motions as a result of input motions) to the actual behaviour exhibited by a proposed mechanism design (actual output motions from the set of input motions).

Gausemeier, et al. [41] and Craig, et al. [22] address the need for integrated mechatronic device models. Schäfer and López [112] define an object-oriented model to define the control capabilities of production resources within a flexible manufacturing cell. The model defines different types of resources with their relating elements and frames including their mathematical representation. Particular emphasis has been given to the modelling of the control aspects of the resources. Seliger and Bollmann [115], Meijer, et al. [80], and Zhang, et al. [143] report function models for the design of devices and systems.

All of these reported models focus only on specific aspects of the equipment characteristics. None of them consider the consequences of modularity in the definition or application of their models. For the fully integrated system development it is however necessary to extend the object focus to include their process relationships. Dori [30] defined an object-process methodology (OPM) that aims to provide a holistic approach to system engineering. It uses objects, processes, and states as basic concepts to describe a wide range of different systems in the business, engineering, and science domain. The proposed methodology is defined at a very high level and is not providing more specific concepts needed for an effective specification of modular assembly equipment.

The focus of an equipment module definition ontology has to be on its process capabilities to allow an effective selection and integration based on required process characteristics (Vos [133]). Several general approaches have been reported for the definition of an equipment module's process capabilities. Zha, et al. [142] use knowledge intensive Petri net for the modelling and analysis of assembly equipment and systems. They stressed the need for further research in the area of linking the process representation with the function structure of the assembly system.

A language representation of function-behaviour-structure for mechanical devices has been introduced by Sasajima, et al. [111] based on ontological engineering principles (Mizoguchi and Kitamura [84]). Their main focus is on understanding the functional capability of devices based on their behaviour and structure. Umeda, et al. [129] and Tomiyama, et al. [124] use qualitative physics to define the relation between structure, behaviour and functions. Sasajima, et al. [111], Mizoguchi and Kitamura [84], Umeda, et al. [129], and Tomiyama, et al. [124] all define behaviour based on physical phenomena.

The reported models are either quite general or focus on quite low level elementary building blocks. The general models make it more difficult to achieve the required level of interpretability while the very elementary models do not take advantage of modular assembly system specific simplifications. There is still a lack of an equipment module specification approach that provides the right level of detail.

2.3.3 Design Frameworks

There are quite a few works looking at assembly modelling and assembly planning (summarised in Homem de Mello and Sanderson [54]). Assembly modelling is generally concerned with defining the relationships between the individual component parts of an assembly. Assembly planning looks at the modelling and selection of best process sequence for a particular assembly problem.

The first step of translation, deriving the required assembly processes from the product representation, has been addressed in a few works. Stadzisz and Henrioud [117] are using a liaisons graph to represent the relationships between the components of an assembly and predicate/event Petri-nets for the representation of assembly process sequences. The link between them is established through an interactive technique that asks the user directed questions.

Rampersad [99] and Delchambre [26] propose integrated methods that cover the whole three phases. Rampersad is giving formal rules and guidelines for the integrated definition of assemblies, assembly processes and assembly systems. Delchambre is introducing a CAD based method for the concurrent definition of assemblies, assembly processes and assembly system. The method, however, stays at a rather abstract level especially on the assembly system side.

De Lit [25] analyses the consequences of designing assembly systems for whole product families as opposed to single products. He proposes a theoretic framework for the concurrent assembly design, assembly sequencing and assembly line design.

Zha, et al. [142] presents a knowledge intensive Petri net framework for the design of automated assembly systems. They are using the formalism to model the whole problem space from assembly over assembly process to assembly system. The framework is intended to assisted assembly system designers in the configuration, simulation and evaluation of assembly systems.

Several other works focus on the designing of assembly systems starting from an assembly process description (Pellichero [96], Rekiek [105], and Vos [133]). Pellichero [96] introduces a framework for the construction of logical assembly line layouts. The method is based on two integrated branches for rough and detailed layout. It involves the selection of assembly methods, line balancing and resource planning.

Rekiek [105] is using a multiple objective grouping genetic algorithm as assembly line design aid. He is applying an equal piles approach to allocate assembly tasks to a defined number of workstations and a branch and cut algorithm for the selection of suitable equipment for each task on the workstation.

Vos [133] is proposing an assembly system development method based on modular assembly equipment. He is analysing the module specification process as well as the module configuration process which eventually yield a complete assembly system. He identifies the matching of process requirements with the capabilities of the modules as one of the most critical problems and uses operations as basis for the matching.

Several design approaches have been developed which demonstrate the principle feasibility of computer aided and knowledge-based assembly systems design (Boër, et al. [11], Bley, et al. [9], Travaini, et al. [125], Zha, et al. [142], Lohse, et al. [76]). The reported approaches only focus on some decision-making

aspects and do not yet address the specific modelling needs for rapid configuration and reconfiguration. Boër, et al. [11] report the results and experiences gained from the development of a computer aided assembly planning tool. The development is mainly focused on gripper selection, sequence generation, and 2D and 3D simulation.

Bley, et al. [9] report a methodology for the knowledge based selection of assembly equipment. They use functions defined in DIN 8593 and VDI 2860 to specify the capabilities of equipment entities. The requirements for the selection of equipment are derived through four types of knowledge: geometry processing, conversion rules, linking rules, and application experience.

Myon-Woong, et al. [86] report on the development of a knowledge based design system for the embodiment design of machine tools. The design process is structured into: user requirements, machine configuration design and analysis (spindle unit, feed drive unit, and constructional elements), re-evaluation of configuration, and detailed design. The design process is supported by a design environment built around a central knowledge base. The reported system architecture consists of seven parts including: the knowledge base, a knowledge manager, an inference engine, a solid modeller, a graphical user interface, a design history manager, and an analysis software.

Lewek [68] proposes a modelling framework for the definition toolbox based manufacturing systems. The model is defined on four levels of representational abstraction: meta-meta, meta, class, and instance level. The model is extending semantic nets as a basic modelling framework. The main focus of the work is to provide a suitable meta-model for the definition of toolbox systems. The approach is not considering some fundamental aspects of modular systems like interface constraints or functional abstraction.

A number of distributed engineering environments have been reported that use mostly agent-based technology to achieve a concurrent integration of all stakeholders in the design process. Rosenman and Wang [109] report a component-based collaborative CAD system for the design of buildings (architecture and construction industry) using an agent-based approach. The reported work is focused on the definition of design components and their application and advantage in a distributed design environment. The paper compares five different system architectures for the

implementation of the design process: integrated mode, distributed-integrated mode, discrete mode, stage-based mode, and autonomy-based mode.

Chao, et al. [16] are addressing communication of design changes in concurrent engineering. They developed an agent-based framework to provide a collaborative environment for communication between design tools, which usually operate in different computer systems. A centralised product model, modelled in STEP, is managed by a design agent. Then mobile agents are managing each design application.

Husslage, et al. [56] propose a framework for simulation-based product design where there are multiple coupled simulation tools and large simulation times. A modelling methodology called Collaborative Meta-Modelling (CMM) is presented. CMM is aiming to combine meta-models for each simulation procedure into one meta-model for the product. The purpose is to simplify and automate coordination of simulation tools.

Chen, et al. [19] present a web-based system for real-time collaborative assembly modelling called e-Assembly. e-Assembly provides concurrent and synchronous design and modelling. Denis, et al. [27] present work from the DIJA project. The project is working on developing a web-based CAD system that is accessible to any user from a simple desktop computer. Wang, et al. [134] present work done on the development of a cooperative design system called WebBlow. The system aims to enable project managers and designers to collaborate over the internet.

A methodology for support in integrated product and process design (IPPD) is proposed by Mervyn, et al. [82]. This methodology is focusing on development of distributed manufacturing applications that are able to support IPPD. Their approach is to use a middleware to ensure a dynamic interface between applications and a common product model. Nahm and Ishikawa [87] describe an integrated product and process design framework for collaborative product design over the Internet. A design model is generated by decomposing the design at three levels: product, process and problem. Li, et al. [69] describe a framework for distributed and collaborative feature-based design.

Xiang, et al. [138] describe agent-based simulation for virtual prototyping of a fluid power system. Domain agents are used to manage components and simulate components' behaviours. Anumba, et al. [5] describe work done in collaborative

design of light industrial buildings. Tang and Wong [121] present a multi-agent framework for control in a computer-integrated manufacturing (CIM) system.

2.4 Knowledge Gaps

Despite the significant developments in the area the reported research does not yet fully address the specific requirements of reconfigurable and evolvable assembly systems on their design and supporting knowledge frameworks. As a result of the exhaustive literature review it became clear that the majority of research in the area of system configuration and design is focusing on the planning, selection, and optimisation of assembly systems on a workstation level of abstraction. There are some reported approaches that look at the more detailed design in some more specific equipment domains. They do, however, not consider the wider environment of the assembly system or do so only in relation to their specific research focus. Figure 2.22 shows an overview of the areas of research.

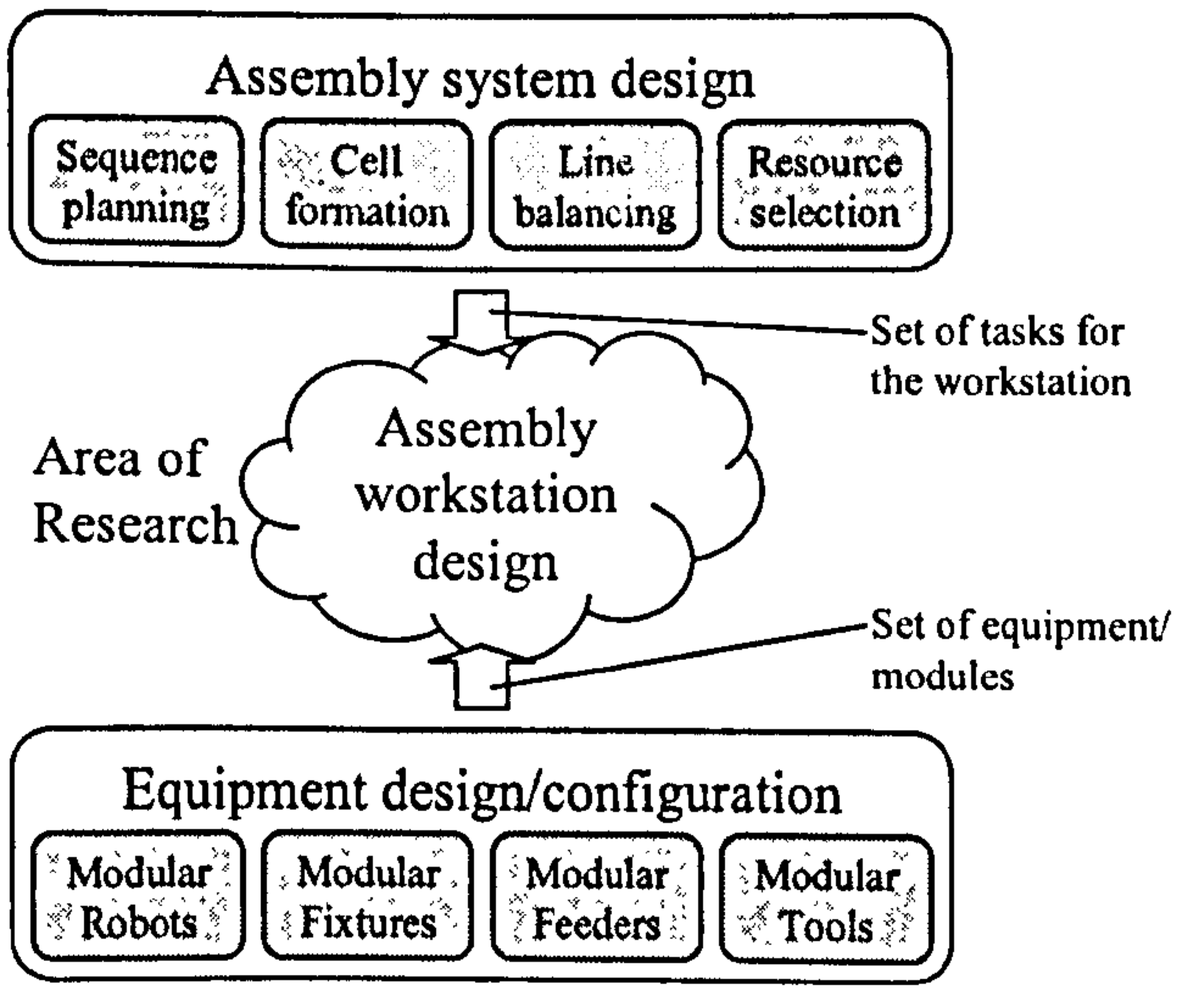


Figure 2.22 Research activities and research gaps in the area of assembly system design

This work is addressing the need for a supporting ontology framework for the requirements driven configuration of modular assembly workstations. The following knowledge gaps have been identified that currently hamper the move towards rapidly deployable manufacturing solutions on demand.

Lack of formal assembly process conceptualisations which would allow a dynamic process specification to the required level of detail while still maintaining a sufficient degree of interpretability to be effective for computer aided design approaches!

For an effective process requirements based selection and configuration of modular assembly workstations it is paramount that the required assembly process can be defined to a high level of detail which can be associated to the available equipment functions. To achieve that within a computer aided environment there needs to be a clear computer interpretable classification of the different process types. Furthermore there needs to be a formalism that defines how currently abstractly defined processes can be decomposed to provide the required higher level of detail. For the requirements driven specification of processes in an integrated framework it is important to provide a clear link between the relevant product characteristics and their enabling assembly processes.

So far no suitable assembly domain specific formal modelling methodology has been reported that can deliver all these requirements. Research effort within the assembly process specification domain has so far been focused on a much higher level of abstraction, assembly task sequence specification, connected with the assembly planning problem (see section 2.3.1). Some general process specification models have been reported that allow higher levels of detail, but these do not consider the specific requirements of the assembly domain. Other research has used very narrowly defined models for specific aspects of the assembly process. Generally there is a lack of a suitable assembly process domain theory.

Insufficient knowledge on equipment models which enable an effective selection and integration of modular assembly equipment solutions based on requirements derived from the product description!

A clear capability model that is linked to the process requirements and that used to synthesise the emergent capabilities of configured workstations is necessary for the effective selection and integration of modular equipment solutions into workstations that can deliver the required assembly capabilities. Furthermore, there should be clear design constraints that reflect the architectural choices of the workstation both during equipment requirements specification and module integration. For effective computer aided specification there should also be a clear classification of different equipment types within a given domain.

For the integration it is vital to understand the structural as well as logical constraints between different equipment modules. So far there is a lack of an assembly domain specific equipment model that defines all the required aspects for an

effective selection and integration of modular assembly equipment. The reported research has either focused on too abstract models or does not taken the constraints of a modular approach into consideration (see section 2.3.2).

Limited availability of an integrated modular assembly workstation definition framework!

Within the reported research there are only very few integrated design methodologies that could address the process requirements driven configuration of modular assembly workstations. Current research has been focusing on the definition of general methodologies for the configuration of modular systems on the machine/workstation level as well as on the specification of modular equipment solutions within individual domains, as for example robotics or feeding.

There are significant cross influences between the assembly process definition and the configuration of the assembly workstation that can deliver the required processes. The relationship between the definition of assembly processes and the configuration of equipment solutions has not been sufficiently explored in the reported research (see section 2.3.3). Current approaches report the cross influences during the definition process but do not address how they can be taken into consideration on a detailed level. Other approaches consider the process and system definition as consecutive steps rather than a concurrent, iterative development.

2.5 Summary

The literature review has given a general background of the main concepts behind this thesis. It includes the principles of modularity, current developments of modular assembly solutions, general design methodologies, and fundamentals of ontologies.

This thesis is addressing the need for new assembly process and equipment formalizations and a method for their integrated definition. The state-of-the-art reported in these areas has been critically reviewed and discussed. This led to the conclusion of the following knowledge gaps:

Lack of formal assembly process conceptualisations that would allow a dynamic process specification to the required level of detail while still maintaining a sufficient degree of interpretability to be effective for computer aided design approaches!

Insufficient knowledge on equipment models that enable an effective selection and integration of modular assembly equipment solutions based on requirements derived from the product description!

Limited availability of an integrated modular assembly workstation definition framework!

In the following chapter the research approach behind this thesis will be discussed. The systematic approach towards addressing the identified knowledge gaps is being described in detail.

3 Research Approach

3.1 Introduction

The vision behind this work is inspired by the need for a dynamic design environment that can facilitate the rapid reconfiguration and evolution of modular assembly systems. The dynamic environment is envisaged to consit of a design space that facilitates the dynamic integration and evaluation of assembly systems in the virtual world. The aim is to allow continuous evaluation and seamless configuration and re-configuration of the assembly system in response to changes in key product, process and system performance characteristics (see Figure 3.1).

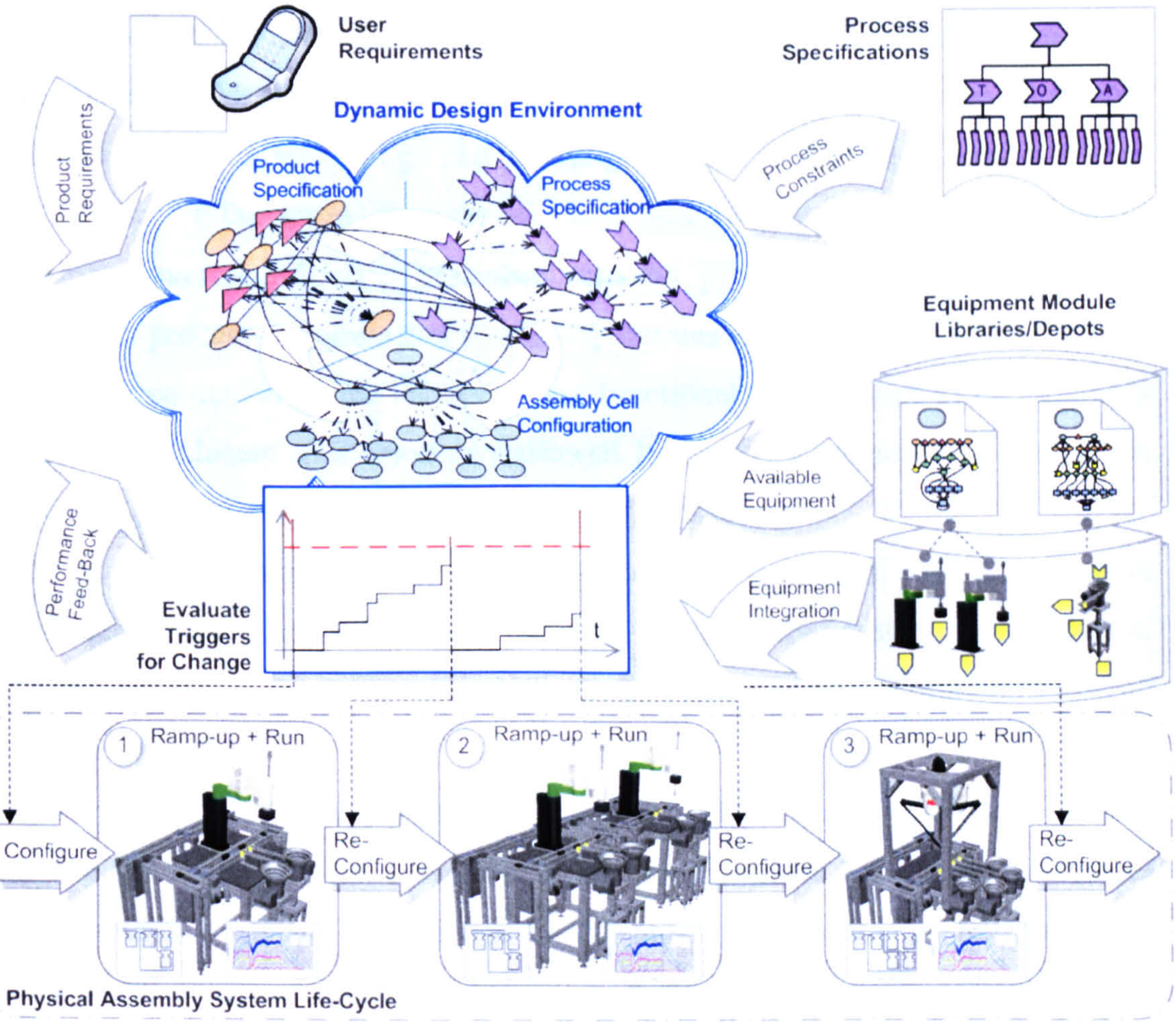


Figure 3.1 Overview of the Integrated Modular Assembly System Design Methodology

The approach should be based on the continuous evolution of the system configuration using the virtual system model and periodically transforming the evolving virtual system into a new configuration of the physical assembly system. This way the assembly system not only responds timely to changes in its environment but also takes advantage of opportunities for reconfigurations arising from new technologies and new equipment solutions becoming available during the lifecycle of the assembly system. The formation and reconfiguration of assembly cells would therefore be performed based on two concurrent processes:

- continuous specification, configuration and evaluation of system solutions using the virtual system model;
- periodic re-configuration of the physical assembly system.

The dynamic environment and the associated methods and models should allow a time-compressed decision-making support that projects the main assembly system engineering activities within a virtual design space to deliver an evolving system design solution. The design environment should combine a set of functionalities such as requirements engineering, process specification, and system design. The environment should also be supported by a knowledge model allowing elicitation, formalisation and reuse of design and planning knowledge. The equipment ontology, which is part of the knowledge model, facilitates structured decision-making for configuring and reconfiguring assembly cells by providing the means to match product and process requirements to the capabilities of different assembly system solutions. Once delivered, the behaviour and functional performance of the assembly cells could be closely monitored and allowed to evolve in response to changes in product and process requirements or system performance.

The system re-configuration process should be conducted by the dynamic environment based on identifying possible triggers for system configuration, modelling responses in terms of system modifications and selecting from among possible alternatives.

The reported work strives to overcome some of the inherent challenges towards achieving the vision of such a dynamic design environment. This chapter discusses the fundamental ideas and assumptions behind the proposed ontology framework for the integrated design of modular assembly workstations and outlines how this work proposes to address the identified knowledge gaps (see section 2.4).

In the following sections a general overview of the assembly system design framework is given followed by the definition of the underlying hypotheses of this work. The assumptions this work has been based on are listed and discussed. The fundamental design activities of a modular assembly system are outlined and linked to the proposed fundamental ontology framework. The chapter concludes with the overview of the fundamental concepts used to define the ontology framework.

3.2 Requirements for the Design of Modular Assembly Systems

It is important to analyse the full lifecycle of an assembly system to understand the requirements for evolvable assembly system solutions and the implied challenges posed for suitable supporting design frameworks and knowledge models. The requirements are very much dependent on the system boundary that is chosen for the analysis. This work is looking at both the lifecycle of a single assembly system and the implications for a domain encompassing design approach. The important difference between the two is that a single system is an instantiation of an existing architecture while a domain-wide approach focuses on the definition of suitable system architectures (Vos [133], Bi and Zhang [7]).

In the following sections the lifecycle of an individual modular assembly system instance will be discussed. The focus will be extended to look at the system design approach for a whole domain. Furthermore, triggers for adaptation (change) will be identified; the different levels of adaptation within a modular system will be outlined; and some mechanisms of achieving the adaptation will be suggested with focus on synthetic design environments.

3.2.1 Modular Assembly System Lifecycle

The lifecycle of an individual assembly system starts with the arising need for an assembly system. A system integrator is normally employed to design a suitable assembly system solution as discussed above. During the design of a modular system, the system integrator selects a suitable architecture, searches for a set of modules that fulfils the requirements, puts them together and installs the system. The system is operated in its current configuration until the need for a change arises that triggers its redesign. A new design cycle is started with the objective to adapt the existing system to the changed requirements. Some of the currently used modules are taken out and

some new ones are added in their place. After the change has been implemented the assembly system is again operated until another need for adaptation arises. This adaptation cycle can be repeated until the chosen system architecture does not yield optimal system configurations for the given sets of requirements anymore.

Figure 3.2 shows a schematic overview of an assembly system life-cycle. The synthetic system design and adaptation need to take place simultaneously to the operation of the system and take the existing structure of the system into consideration.

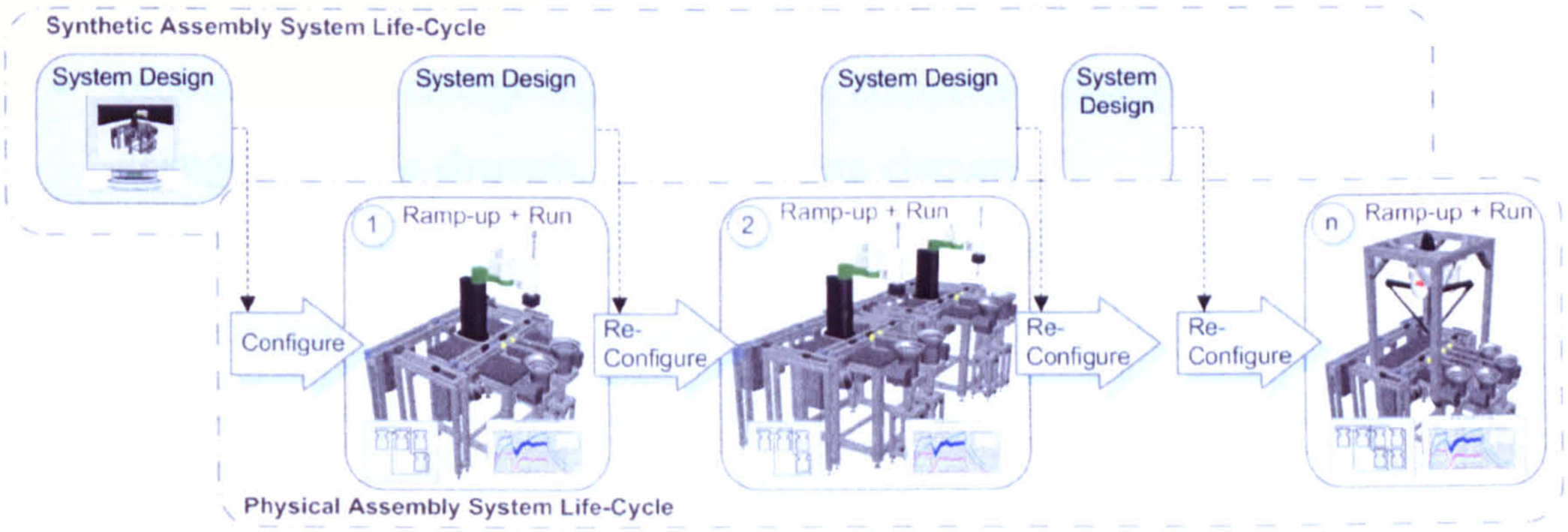


Figure 3.2 Overview of a representative evolvable assembly system lifecycle

3.2.2 Domain-Wide System Architecture Development

The definition of system architectures is generally closely related to the maturity of a domain. Once the processes used in a specific domain have reached a certain level of stability or maturity, it becomes desirable to consolidate the knowledge that has been developed in this domain. One of the vehicles to achieve this is through common architectural guidelines and rules.

The definition of a system architecture starts with an analysis of the requirements posed in a given domain. The range of process requirements for the domain needs to be defined and turned into a suitable architectural system framework consisting of abstract module definitions with basic process capabilities and configuration guidelines. Equipment manufacturers can use the module definitions to design their own specific module instances. These modules can then be used by the system integrators to define specific assembly system solutions. The definition and evolution of assembly systems can take place within the boundaries of the system architecture. This however is the case only as long as modules developed within the architecture support a significant proportion of the domain requirements. At some point the domain requirements will have shifted significantly enough to make adaptation of the

architecture necessary. This requires a new assessment of the domain requirements which will highlight the required changes.

3.2.3 Adaptation Triggers

An adaptation is always the result of changes in the external environment or due to current suboptimal characteristics of the system; these are called trigger conditions. They generally need to exceed a certain threshold before they require the system in question to adapt. The synthetic assembly system design environment needs to be able to address the flowing sources of change:

- product related change triggers include: component changes, product structure changes, volume changes, and assortment changes
- process related change triggers include: new available processes
- system/equipment related change triggers include: new modules have become available, new architectural guidelines have become available, and actual system feedback indicates suboptimal operational performance

3.2.4 Levels of Modular System Adaptation

A modular assembly system provides different mechanisms to accommodate requirement changes. They can be facilitated through adaptation on three levels:

- Level 0: Parametric changes – adapting the behaviour of available capabilities; e.g. changing the force settings of a pressing device
- Level 1: Logical changes – adapting the utilisation of available capabilities (skills); e.g. change of process sequences from one product to another
- Level 2: Structural changes – adapting the available capabilities; e.g. changing one process module for another one or adding an additional assembly station

Level 0 adaptation is the easiest and the least powerful. Level 2 adaptation requires the highest effort but allows for the highest impact of change. Figure 3.3 shows the principle levels of adaptation in a modular system. A configured modular system has a number of equipment modules that have a number of skills which define their assembly process capabilities. The equipment modules are physically connected to each other on level 2. The overall process capability of the system is defined through the logical relationships of its module skills on level 1. The specific process behaviour is defined through the parameter settings on level 0.

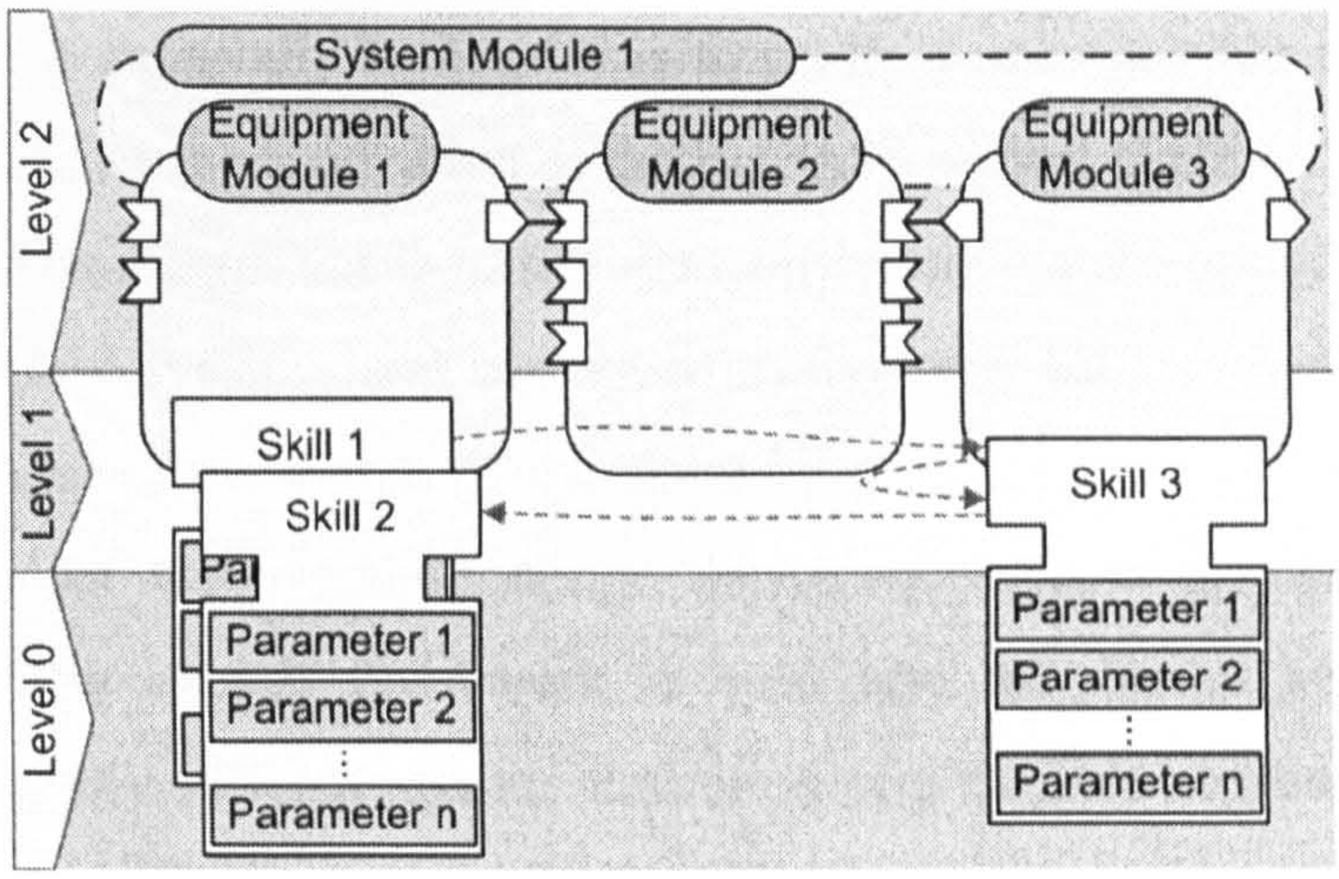


Figure 3.3 Levels of adaptation in a modular assembly system

3.2.5 Mechanisms for Adaptation

Different mechanisms to facilitate adaptation are required for the different levels of system change. The specific focus of this work is not so much on the technical realisation of the control level but rather on the requirements that result for the design and re-design of an assembly system. On level 0 for example the parameters need to be initially set by the design environment but the later adaptation should occur at the actual system by the system operator. On level 1 the logical structure should be adjusted automatically by the control of the system. The design environment only sets the boundary for the possible runtime changes. This boundary defines the inherent flexibility of the assembly system. The main design effort has to be directed towards the definition of the system capabilities in their subsequent adaptation on level 2. These mechanisms will be discussed in more detail in the next section which is focused on the discussion of a design framework for modular assembly systems to achieve reconfigurable and evolvable assembly systems.

3.3 Overall Modular Assembly System Design Framework

The aim behind modular assembly systems is to design a set of equipment modules that can be combined to deliver a wide range of different processes. The success of modular system solutions depends to a large degree on the completeness and acceptance of their system architecture (Pahl and Beitz [94]). The current practice in assembly system design is for different system integrators to define their own more or less rigid proprietary system architectures. This is a very effort and know-how

intensive process especially if the aim is to create a domain-wide open architecture which is for instance the motivation of the EUPASS project (EUPASS [33]).

The design framework needs to simultaneously address the design and redesign of modular assembly systems and the specification of a suitable system architecture definition to achieve the vision of dynamically evolving assembly system solutions. Figure 3.4 shows the principle design activities required in such an integrated environment. The overall framework is split into the design and adaptation of individual assembly systems and the definition of a suitable architecture (Vos [133] and Bi and Zhang [7]). Both the actual assembly system design and the architecture definition have to go through the product, assembly process, and assembly system domains as defined by Rampersad [99].

The purpose of the assembly system architecture definition is to define a set of guidelines and specifications that enable equipment modules to be designed in such a manner that they can be integrated to form wider system solutions. The architecture definition should also make the constraints arising from the set of available equipment modules accessible as constraints during the early stages of the assembly system design process.

The purpose of the assembly system design is to find a suitable solution for a given set of product based assembly requirements by selecting and configuring available equipment modules into an assembly system.

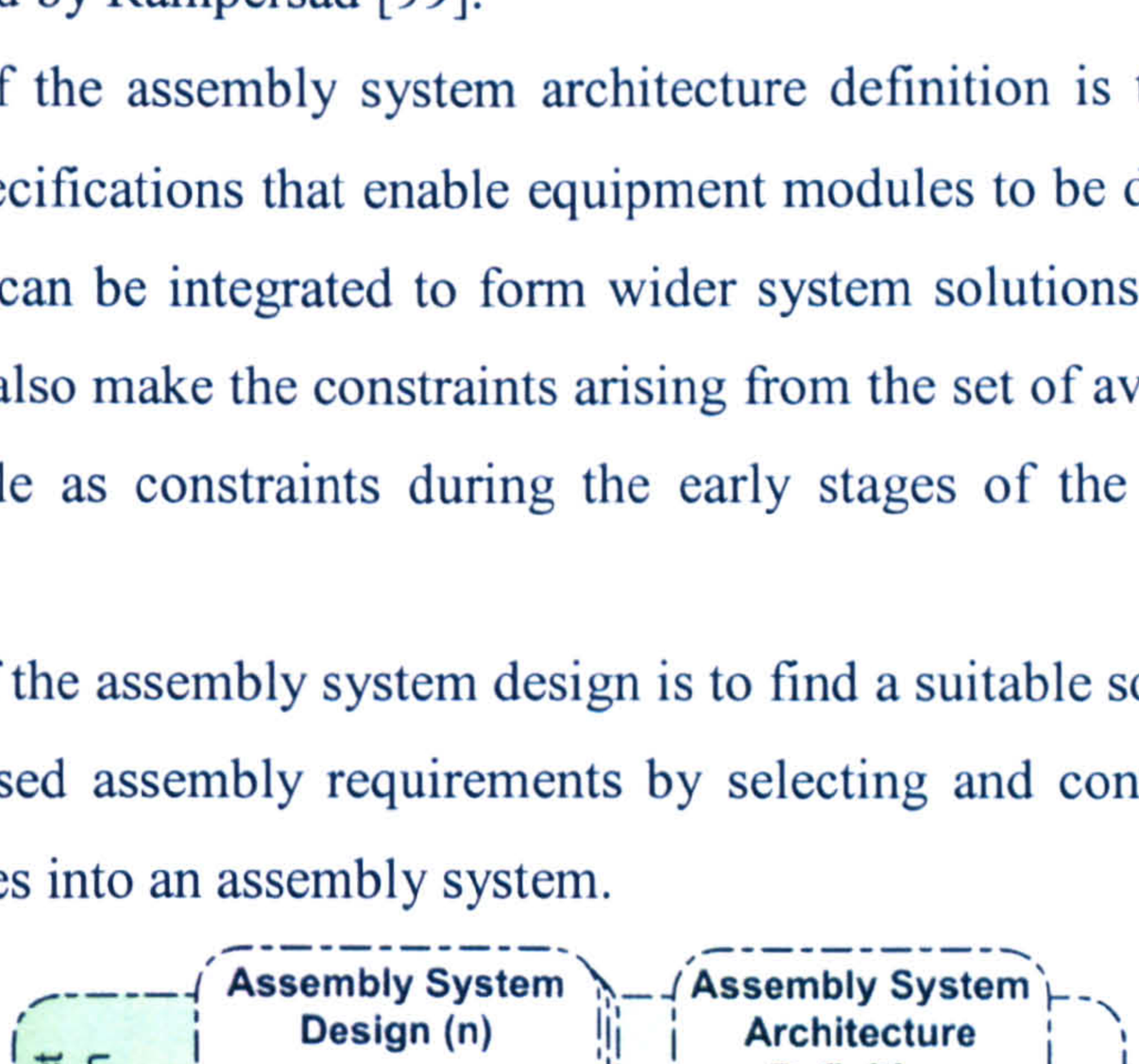


Figure 3.4 Overview of the fundamental design framework

Both of the assembly system design and the architecture definition activities have a definition and analysis phase. The definition phase translates requirements into solutions through a transition between the product, assembly process, and assembly system domain whilst the analysis phase validates and verifies that the proposed solutions actually match the original requirements. Each definition phase is defined in a succession of iterative loops between the product, process, and system domain. Figure 3.5 shows the definition process on the right hand side and the analysis process on the left hand side. Each transition has a definition, analysis, and validation phase.

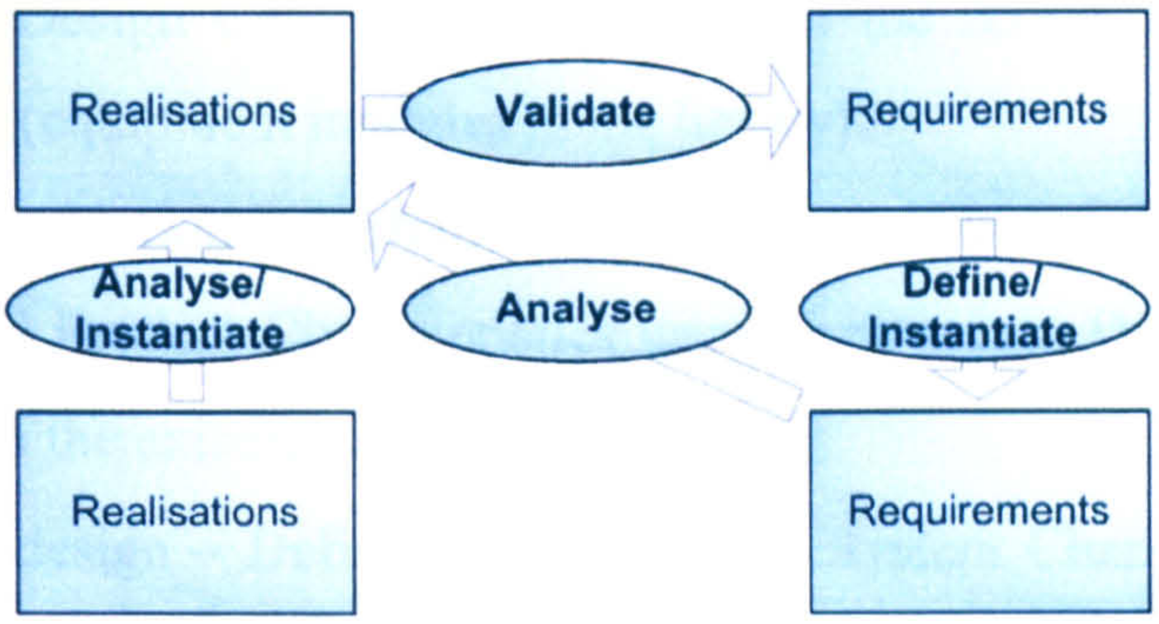


Figure 3.5 Fundamental design activities

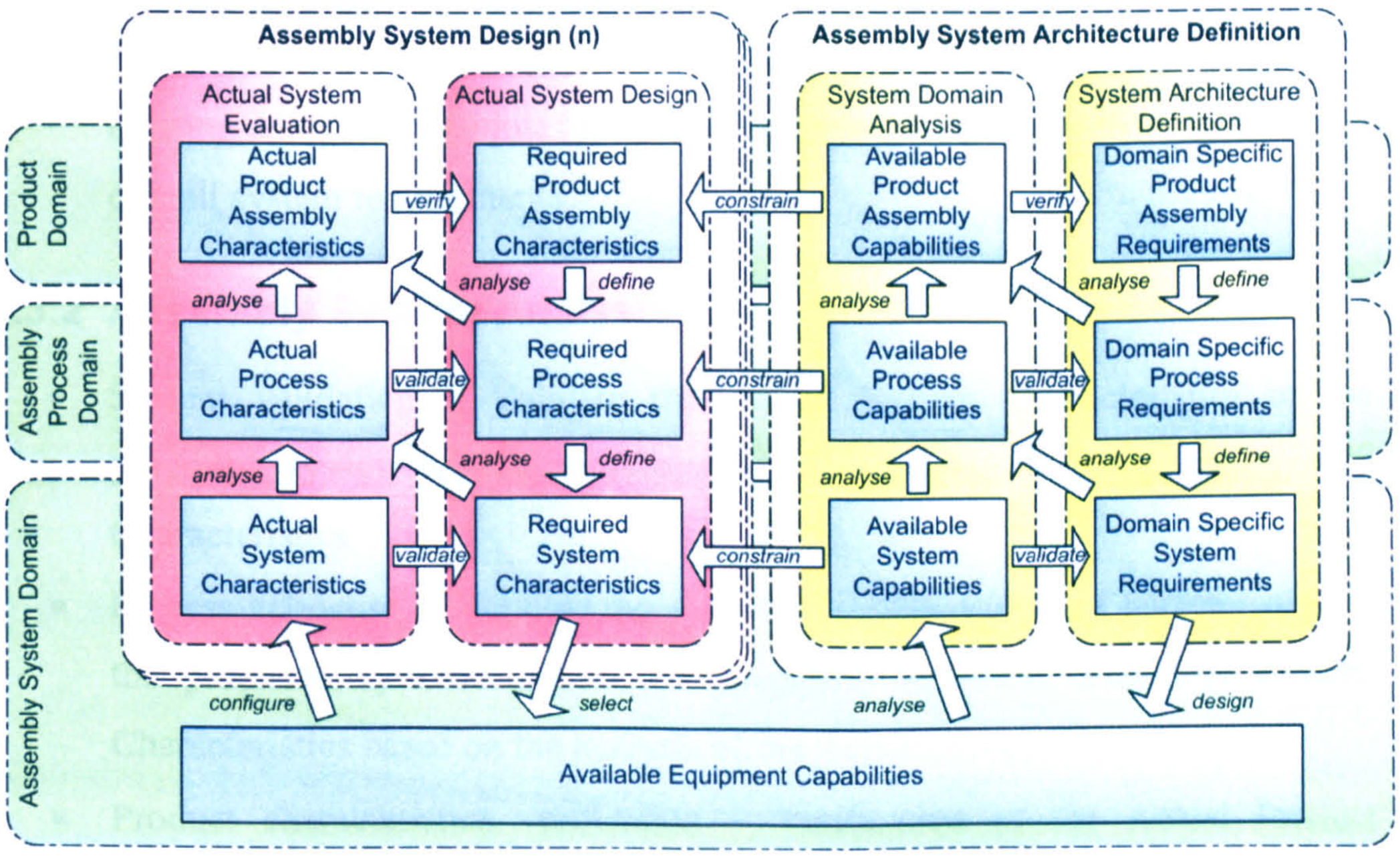


Figure 3.6 Integrated Design Framework for Modular Assembly System and Architecture Design

Figure 3.6 shows the overall integrated framework for the design and specification of assembly systems and their architectures. An ontology framework that aims to support the full design process of modular assembly systems has to provide suitable formalisms to capture the concepts required and defined during the different design

activities. It would also need to maintain the relationships between the concepts to maintain the consistency of the model and allow an easier change of existing system configuration. The design activities will be described in more detail in the following sub-sections.

3.3.1 Assembly System Design

- Product/Project definition – Definition of the Required Product and Project Characteristics based on the requirements of the product designer and guided by Product Design Constraints derived from the actual available Assembly Capabilities (equipment modules in the library).
- Process specification – Definition of Required Process Characteristics from the Required Product Characteristics using the Process Definition constraints derived from the existing equipment modules.
- Conceptual design – Definition of Required System Characteristics from the Required Process Characteristics using the System Design Constraints derived from the existing equipment modules.
- Equipment selection and system configuration – Finding and integrating existing equipment modules into assembly system solutions that can fulfil the overall system requirements.

3.3.2 Assembly System Analysis

- System validation – Validate the Actual System Characteristics of the proposed system specification against the original Requirement System Characteristics.
- Process validation – Validate the Actual Assembly Process Characteristics of the proposed systems against the originally Required Assembly Process Characteristics based on the analysis of the Actual System Characteristics.
- Product characteristics verification – Verification of the Actual Product Characteristics that can be achieved with the proposed system against the originally Required Product Characteristics based on the analysis of the Actual Process Characteristics.

3.3.3 System Architecture Definition

- Domain-specific Product Requirements Definition – Analyse the requirements posed on assembly systems from a specific product domain.
- Domain-specific Process Requirements Definition – Define the assembly process capabilities required by a domain based on its required product characteristics.
- Domain-specific Assembly System Architecture Definition – Definition of the fundamental assembly system structure required to achieve the process capabilities required by the domain.
- Equipment Module Design – Design of Equipment Modules that fulfil some of the requirements defined by the Domain Specific System Architecture.

3.3.4 System Domain Analysis

- Domain-wide System Configuration Validation – Analysis of the existing Equipment Modules to determine the degree to which the proposed System Architecture has been created.
- Domain-wide Process Capability Validation – Analysis and abstraction of the process capabilities that are available within a domain based on the current System Design Constraints.
- Domain-wide Product Realisation Verification – Verification of the currently existing Product Design Constraints against the original Product Domain Requirements based on the analysis and abstraction of the currently existing Process Definition Constraints.

3.4 Research Methodology

The work is aimed towards achieving a holistic domain theory for the design of modular assembly workstations which is thought to be one of the fundamental requirements for truly evolvable assembly systems to succeed.

To achieve this aim it would be necessary to create a complete implementation of the whole theory and carry out substantial validation work across the whole domain. The work involved to create a complete domain theory goes far beyond the scope of the reported research. The proposed ontological framework is not intended to provide

a complete domain theory but rather to build a suitable foundation and trigger further development. The specific challenges that need to be overcome have been outlined in the discussion of the knowledge gaps (section 2.4).

The focus of this work is primarily on the development of a suitable methodology for the design of modular assembly workstations. It has been assumed that an architecture definition process has been carried out by domain experts. Only the formalisms to capture the results of the architecture definition process are being considered as part of this work. The aim is to make the knowledge of the domain experts available during the system design process.

3.4.1 Research Objectives

The identified knowledge gaps have been turned into the following list of research objectives. All objectives are directed towards enabling effective sub-workstation modular equipment configuration.

- Definite a new assembly process domain ontology for the specification of the required process capabilities at the right level of detail. The definition focuses on the following aspects to address the identified knowledge gaps:
    - Suitable topological structures for a dynamic definition of assembly processes
    - Classification definition of assembly process types based on the liaison characteristics they enable
    - Generic definition of hierarchical decomposition constraints
- Develop a new assembly equipment domain ontology for the process-requirements-driven selection and integration of modular assembly workstations. The development focuses on the following aspects to address the identified knowledge gaps:
    - Suitable connection formalism to allow easy integration of modular equipment components into a composed workstation
    - Process capability-based classification definition of equipment types
    - Generic definition of architectural design constraints
- Formulate a new method for the integrated process specification and assembly workstation configuration using the developed process and equipment domain ontologies. The formulation of the design approach focuses on the following aspects to address the identified knowledge gaps:

- Method for the decomposition and specialisation of assembly tasks
- Method for the matching of required process capabilities against existing hardware capabilities
- Method for the integration of equipment modules into assembly workstation
- Method for the synthesis of assembly workstation process capabilities

3.4.2 Hypotheses

This section outlines the hypotheses which were defined at the outset of this work as potential solution approaches that overcome some of the challenges for the rapid design of modular assembly systems. They build the foundation for the reported research work and stand at the core of the proposed framework definition. The aim is not to establish an absolute proof or disproof of this hypothesis but to provide a foundation for their more elaborate exploration and criticism. This work provides a more detailed elaboration of the implications of the hypotheses and a first critical discussion of their merits. Figure 3.7 shows an outline of the theories and paradigms the work has been based on.

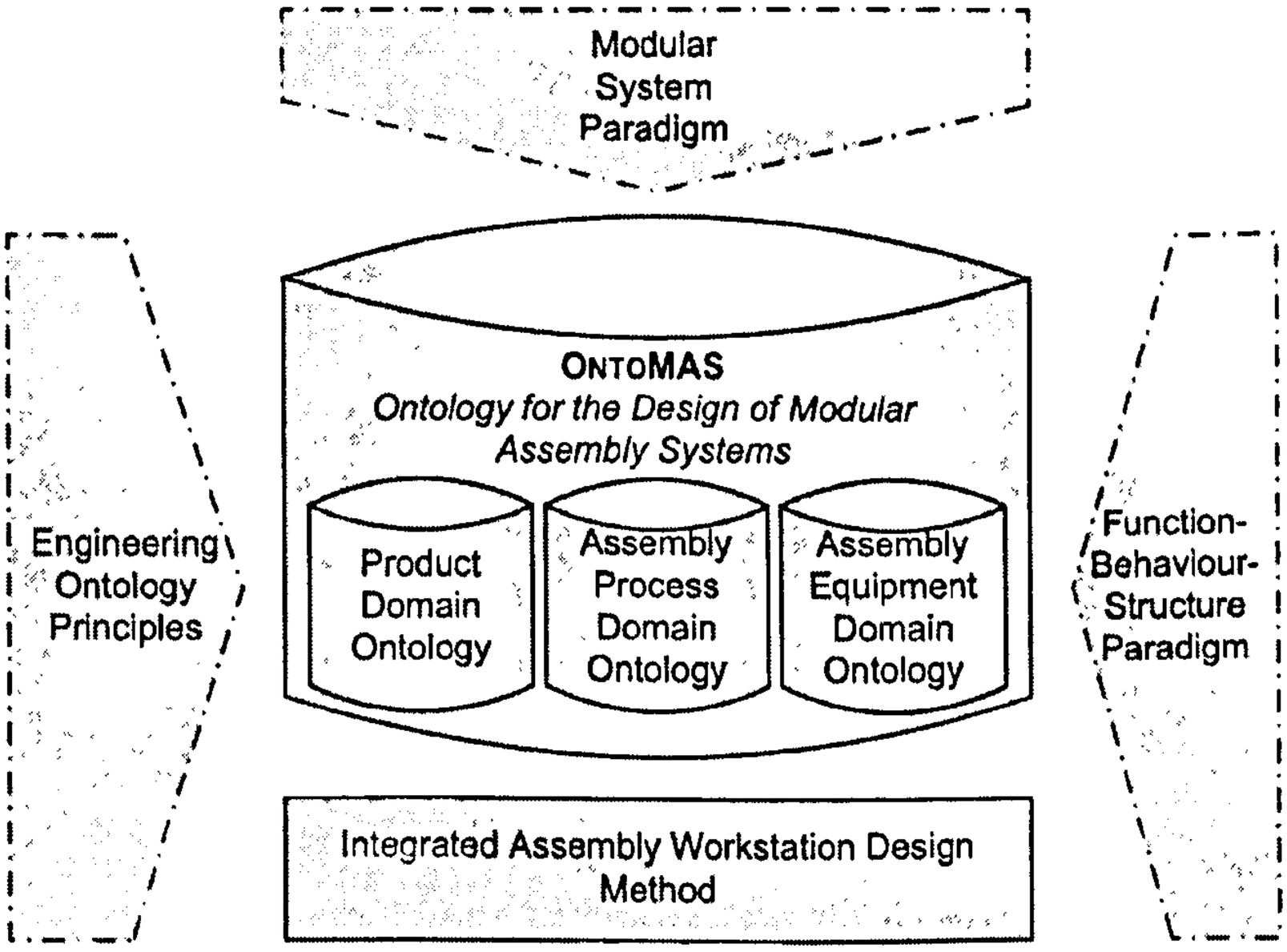


Figure 3.7 Fundamental Research Approach

Use of taxonomies to capture the meaning of the different domain concepts

At the outset of the work it was hypothesised that it should be possible to define a set of elementary activities, derived from the equipment domain, that can be combined to define the majority of the required assembly processes in a specific

domain. The expected benefit of this approach is an approved suitability of the process model for computer-based interpretation and reasoning.

Definition of fixed hierarchical specification structures to address the inherent complexity of the design problem

If both the assembly process model and the assembly equipment model are defined on fixed hierarchical levels it should be possible to define a clear relationship between them. This should significantly reduce the selection and validation effort of equipment configurations since they can be independently compared on different levels of hierarchy.

Capturing of reoccurring design patterns in predefined concepts

If both the process model and equipment function model use the same fundamental levels of hierarchy and have a defined relationship between their classifications, it should be possible to create process composition patterns that can be used to simultaneously guide the decomposition of processes and synthesis of equipment functionality. The synthesis of functional equipment capabilities essentially becomes the inverse transformation of the process decomposition through the defined relationship between them. This relationship is expected to significantly reduce the process specification and functional capability-synthesis effort. Furthermore it is expected to improve the equipment selection and validation mechanism.

Application of the function-behaviour-structure paradigm to enable effective equipment selection

The application of the function-behaviour-structure paradigm should make it possible to define equipment characteristics at a level of abstraction that is suitable for their selection, configuration, and evaluation. This would be a significant step towards the use of vendor independent equipment definitions.

Use of predefined concepts to capture the specific constraints of modular systems

The integration of some of the fundamental principles of modular equipment solutions into their models is expected to improve the configuration behaviour of the

model. The equipment description should only expose the characteristics that are immediately useful for its integration into the design framework, no more and no less.

3.4.3 Approach

The research approach has been structured into three concurrent phases: requirements analysis; detailing of the proposed approach; and initial verification. Figure 3.8 shows an outline of the research approach and how it is reflected in the structure of the thesis.

The motivation and requirements for the ontology framework proposed in this thesis where derived from an exhaustive literature and state-of-the-art review (see chapter 2). The reported research in the areas of product, assembly process, and equipment specification was carefully reviewed. Integrated specification and design frameworks were analysed for their applicability. However, despite the significant work in the area there is still a lack of holistic specification and design frameworks that can support the specific needs of the whole modular assembly system design process. Furthermore, specific industrial requirements were derived from informal interviews and meetings with relevant industrial partners. They supported the need that was established from literature.

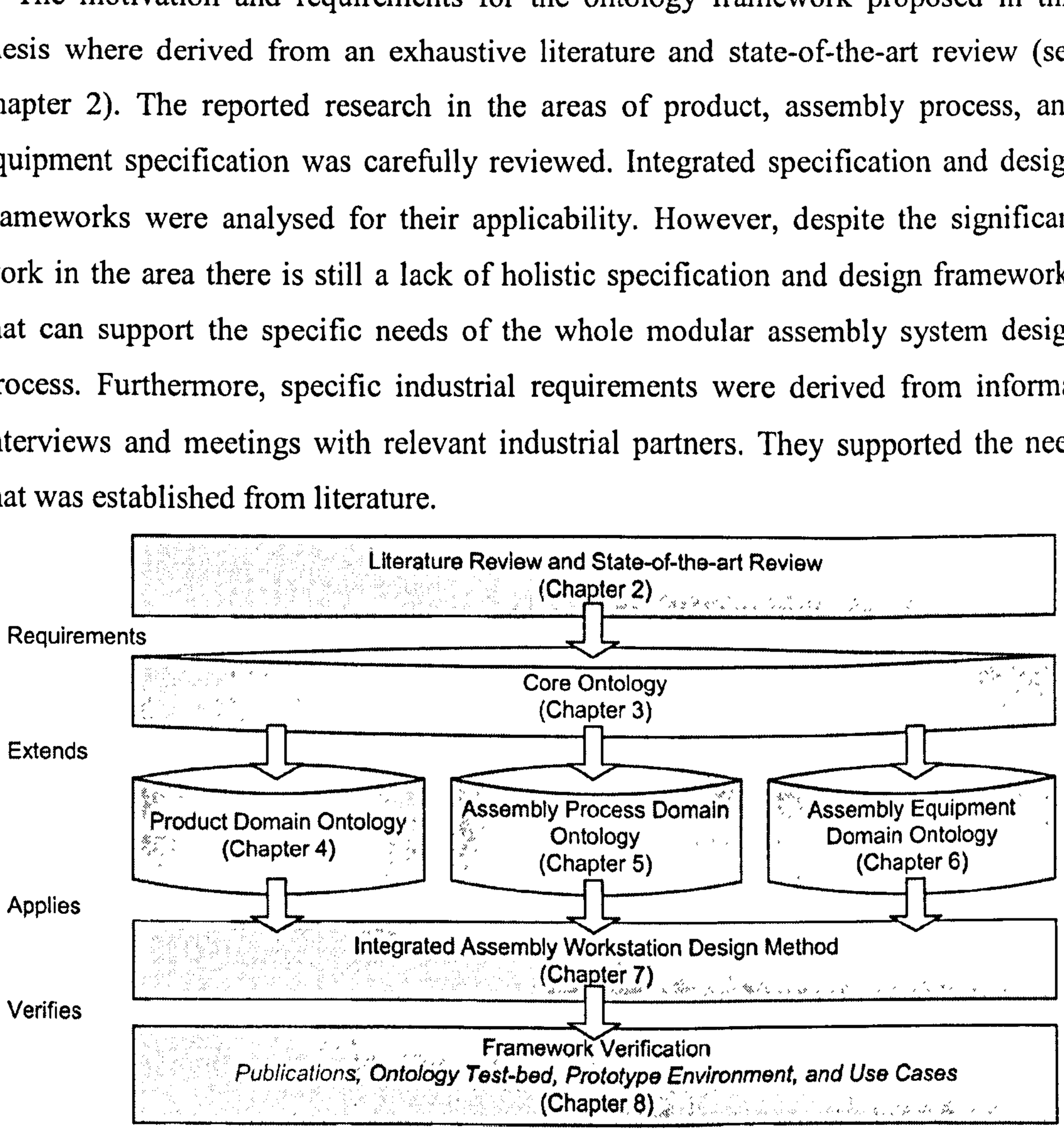


Figure 3.8 Systematic Structure of the Research Approach

The ontology engineering paradigm has been found to be the most promising approach towards a common modular assembly system domain theory that can successfully support the needs to the domain. A core ontology framework has been

developed that provides the fundamental formalisms for an integrated design support framework (see section 3.5). This ontology has been extended to capture the specific needs of the product, assembly process, and assembly equipment domains (see chapter 4, 5, and 6 respectively). The new developed domain models have been applied in an integrated assembly workstation design method (see chapter 7). The ontologies and design methods are specified using mainly CommonKADS visual notations (Schreiber, et al. [114]).

The results of the work conducted as part of this thesis have been reported on a number of international conferences: IEEE International Symposium on Assembly and Task Planning (Lohse, et al. [70], Lohse, et al. [73]), International Precision Assembly Seminar (Lohse, et al. [76], Lohse, et al. [74], Hirani, et al. [52], Ratchev and Lohse [101]), International Symposium on Robotics (Ratchev, et al. [100], Ratchev, et al. [103]), and CIRP International Conference on Reconfigurable Manufacturing (Lohse, et al. [71]). Only fully refereed conferences were considered for publication. Furthermore, the work has been submitted and published in the journal of Assembly Automation (Lohse, et al. [75], Ratchev and Lohse [102]) and accepted for publication in the International Journal of Flexible Manufacturing Systems (Lohse, et al. [72]).

The proposed new ONTOMAS framework has been fully instantiated in Protégé, an ontology definition framework (Protégé [97]), to test its general technical soundness. The Protégé Axiom Language (Grosso [45]) was used to test the verifications of models based on the proposed design patterns. Furthermore the JESS (Friedman-Hill [38]) plug-in for Protégé was utilised to establish the appropriateness of the proposed formalisms in reasoning applications.

A prototype distributed decision-making environment has been developed which demonstrates the general applicability of the proposed framework. The proposed framework and approach have been applied in several industrial and synthetic use cases which provided a first verification. Finally, the proposed ONTOMAS framework is being applied in two major European projects (EUPASS [33], E-Race [32]).

3.5 Ontology Framework (ONTOMAS)

One of the crucial factors for the success of the proposed modular assembly workstation design framework is a well defined model that supports the decision making process. The model needs to be standardised in order to exchange information

and knowledge between different stakeholders, but at the same time it needs to be extendable to deal with future requirements and to provide customisation to cater for the specific needs of the different stakeholders. Furthermore, the model needs to be equally suitable for human as well as machine interpretation since the decision makers could be either or both. There is also a specific need for the model to cater for incomplete and not fully defined specifications.

"Ontology is concerned with the study of being or existence and their basic categories and relationships, to determine what entities and what types of entities exist. Ontology thus has strong implications for conceptions of reality" (Wikipedia [136]).

This definition is supported by Gruber [46] who defines ontology as: "an explicit specification of a conceptualisation". The motivation behind defining ontologies is that they are trying to capture not only the vocabulary of a domain but also their intended meaning (Chandrasekaran, et al. [15]). An ontology defines the concepts of the domain, their relationships, and their attributes. The representation of the ontology is in most cases independent of its actual instantiation in an application environment. This provides the opportunity to define a holistic model that can bridge traditional boundaries between domains.

This section gives an overview of the proposed fundamental structure of the proposed ontology framework. The purpose of the ontology framework is to support the integrated design of modular assembly systems and to facilitate the move towards more responsive assembly solutions. The ontology is called ONTOMAS which abbreviates its purpose (**Onto**logy for the design of **M**odular **A**ssembly **S**ystems).

ONTOMAS follows the same fundamental structure as the overall modular assembly system design framework discussed in section 3.3. It caters for the needs of the different design activities within the modular assembly system domain by providing different models that capture the results of their knowledge transformations (see Figure 3.7). The relationships between these models have been unified to provide a clear definition across the whole domain.

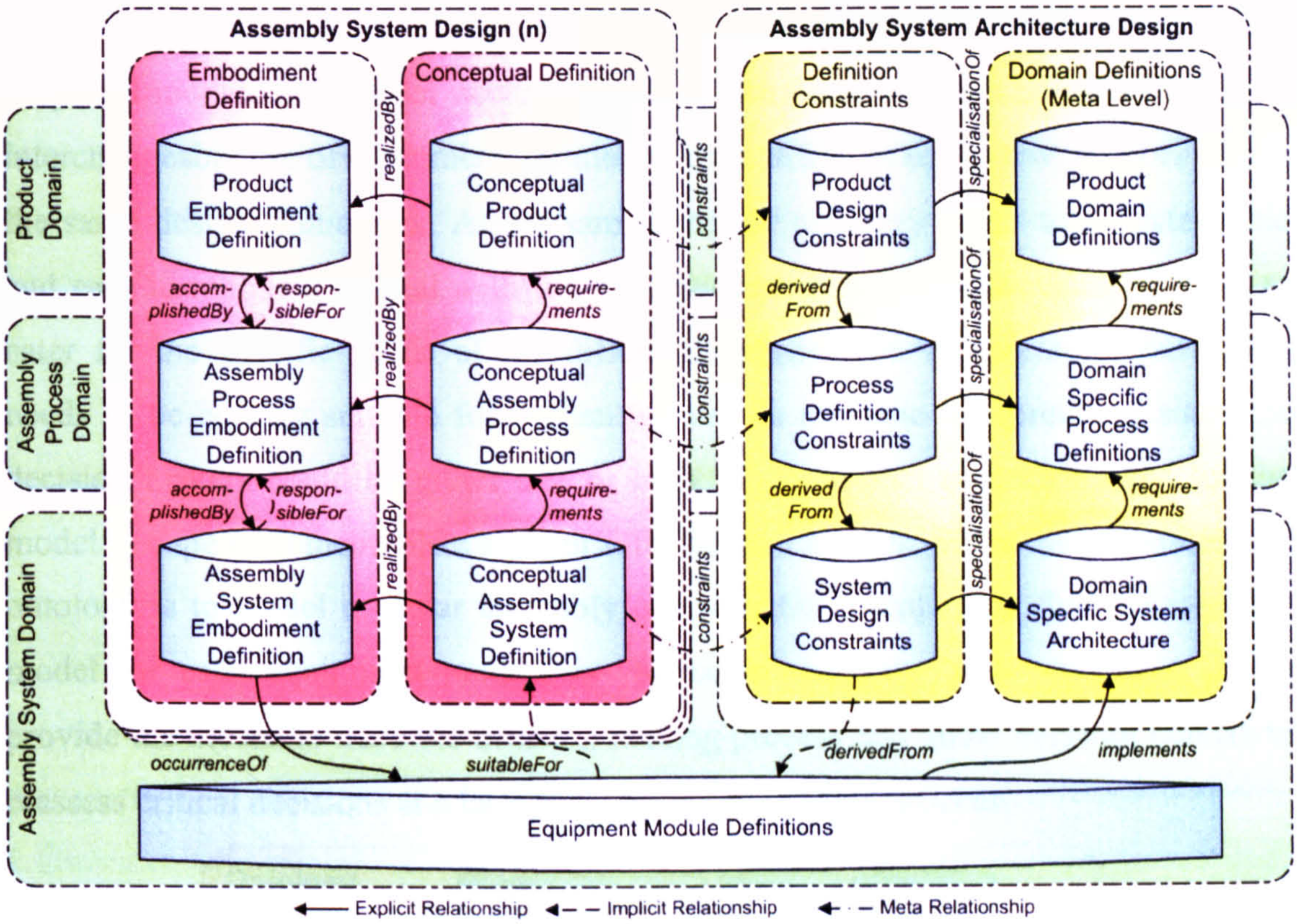


Figure 3.9 ONTOMAS Domain Knowledge Overview including principle Relationships

The causal relationships established through the definition processes are defined with the <requirements> relationship. The causal relationships resulting from the analysis process during assembly system embodiment design are defined by the <accomplishedBy> relationship. On the assembly system architecture side, this association is established through the <derivedFrom> relationship to highlight the constraint nature of the domain.

The association between the concepts of the system design domain and of the system architecture definition domain are defined through the <constraints> meta-relationship. The current definition constraints define a subset of the domain definition. This causality is expressed through the <specialisationOf> relationship. The embodiment definitions of the system design constitute solutions for the conceptual definitions. Their association is defined through the <realisedBy> relationship.

3.5.1 Ontology Structure

One of the crucial factors for the success of the proposed ONTOMAS framework is its ability to provide the information needed to support the entire design decision making process. However, in addition to the information content requirements, the

structural implementation of the ontology is a critical aspect for the success of the proposed model. The model needs to have a high level of standardisation to allow interchangeable use of equipment definitions from different equipment vendors within the same design framework. At the same time the ontology needs to be extendable and easy to maintain to deal with future requirements, and provide customisation to cater for the specific needs of the different stakeholders. Furthermore, the model needs to be equally suitable for human as well as machine interpretation since the decision makers could be either one or both. There is also a specific need for the model to cope with incomplete and not fully defined models. Since the purpose of the ontology is to model modular assembly equipment it is important that the resulting model for each equipment module is self contained. Finally the model needs to provide the means to trace the decision making process and allow decision makers to reassess critical decisions at a later stage during the design process.

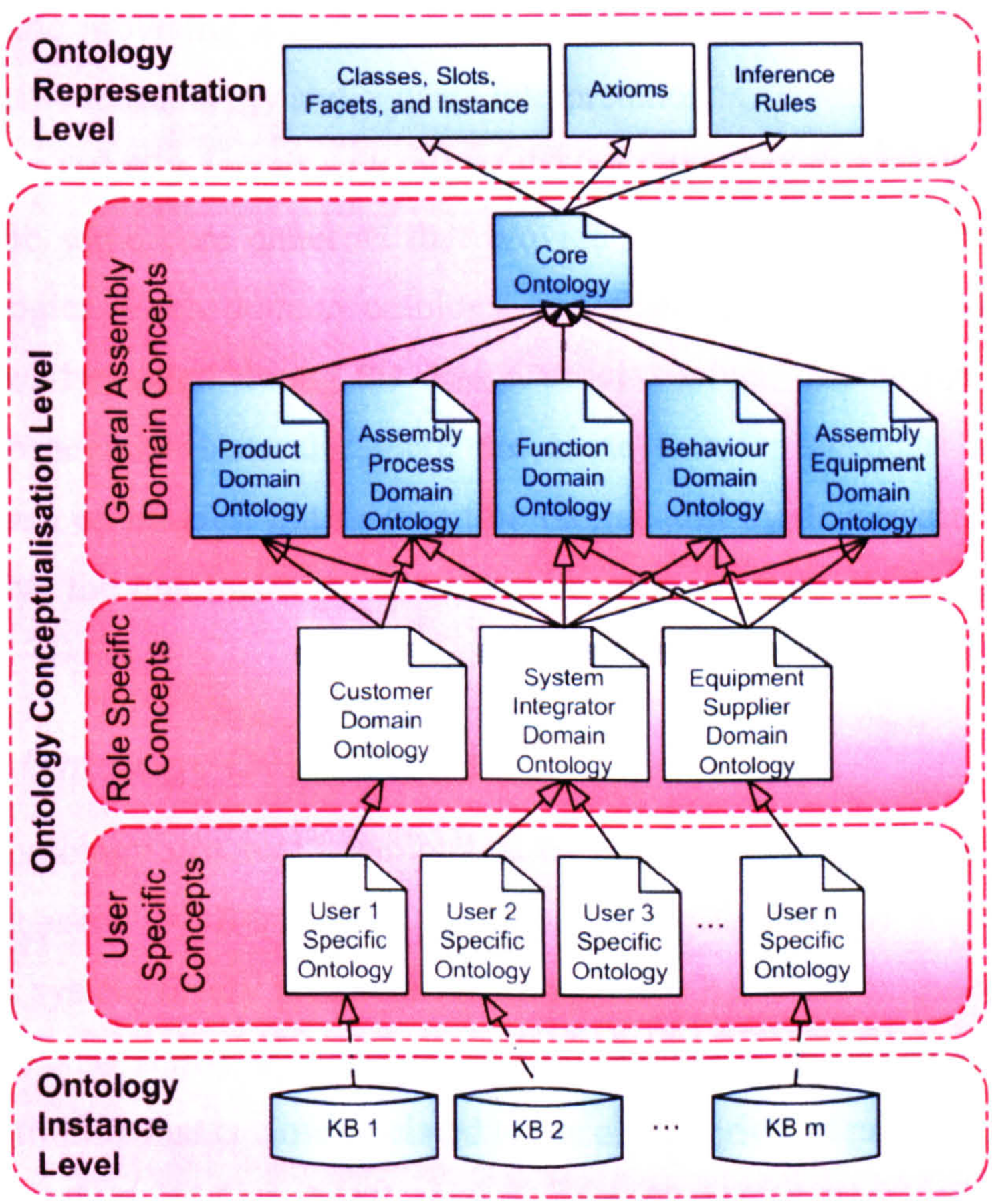


Figure 3.10 Overview of the Modular Ontology Structure

Ontologies generally have three representation levels (Daconta, et al. [23]): the underlying ontology representation level, the ontology conceptualisation level and the

ontology instantiation level (see Figure 3.10). The ontology representation level defines how the different concepts, attributes, constraints and rules, which are used to describe the concepts of the ontology, are implemented. A frame based knowledge representation has been chosen to define the concepts and their attributes in an object oriented manner using UML related notations throughout this work. The specific model constraints are expressed as axioms and the design decisions are modelled as inference rules.

On the ontology conceptualisation level all the domain specific concepts, attributes, constraints and rules are defined. The ontology definition has been modularised and split into more specific domain ontologies to improve the maintainability of the ontology.

The domain ontologies are divided into generic, role, and user specific concepts definitions. This allows all the design decisions to be based on generic concepts while at the same time providing a mechanism to enable different stakeholders to define their own specific terminology and concept interpretation.

The generic assembly domain ontology is the main focus of this work. It is divided again to define some core concepts that provide the foundation for more specific domain ontologies. Each domain ontology is defined as a separate aspect of the knowledge transformation during the design process which can be maintained and updated independently. Naturally there are some interdependencies between the different domain ontologies, but they can be limited and by doing so make it much easier to evolve the ontologies by incorporating new concepts and attributes in the future.

3.5.2 Core Ontology Overview

The basic ontology structure is defined based on the general ontology engineering principles suggested by Borst, et al. [12] providing formalisms for aggregation, topology, and system theory principles. This structure has been extended to include formalisms for abstraction, occurrence, and the definition of design patterns. This results in a definition that is closely related to the object-oriented paradigm.

The core ontology of ONTOMAS provides some basic relationships and concept types that are common across all the other, more specific, domain ontologies. These basic concepts and relationships reflect the fundamental needs of engineering design processes. Engineering design processes are fundamentally solution finding processes

(Pahl and Beitz [94]) which have to address the following aspects: specification of the requirements, search for solutions, evaluation of available solutions, and selection of the best suitable solution. These aspects can be recognised in the design framework proposed in section 3.3 above. Another aspect of the engineering design process is that it needs to deal with highly complex and incomplete problem and solution definitions. Hierarchical problem decomposition and solution synthesis approaches are generally used to address this inherent complexity.

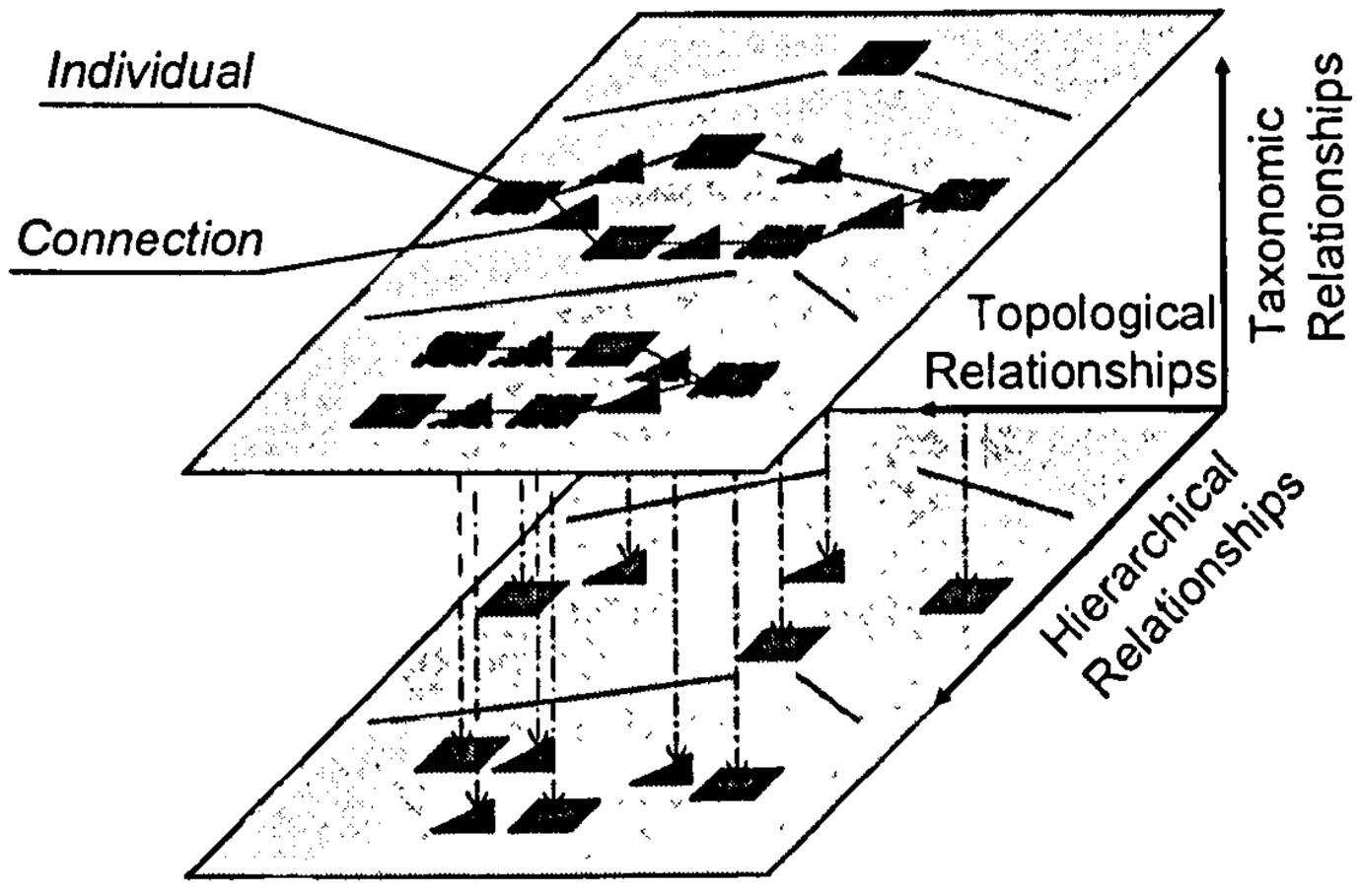


Figure 3.11 Fundamental Relationships between the ONTOMAS concepts

ONTOMAS defines some basic relationships which address the requirements arising from the engineering domain. They provide formalisms for decomposition, specialisation, and interrelationships between concepts as suggested by Pahl and Beitz [94]. Figure 3.11 maps the basic relationships on three axes to demonstrate how they define the fundamental design space. A more detailed description of those relationships is given below:

- **The hierarchy relationships** are based on the principles of mereology. Mereology is the formal study of the relations between parts and wholes (OED [90]). The hierarchy relationships provide the mechanism to deal with the highly complex domain model instantiations. By defining different levels of detail it becomes much easier to define and understand complex models. The hierarchy is defined by grouping lower level concepts and linking them to a representative higher level concept.
- **The topological relationships** define how different concepts are related to each other. Topology is define as "The way in which constituent parts are interrelated or arranged" by OED [90]. The topological relationships include for example the connection between equipment modules or the temporal

relationships between assembly activities. The topological relationships need to express the characteristics of the connections, for example, that the connection between two equipment modules could be dynamic or static. Structure relationships are therefore modelled as concepts in their own right.

- **The taxonomic relationships** specify the classification hierarchy between the different concepts. These relationships are defined using a super/sub-class structure and allow a gradual specification of more and more concrete models. For example at an early design stage it might only be known that a feeder will be required but not yet what specific type.

ONTOMAS provides some core concepts additionally to the basic relationships. They provide the formalisms to specify requirements, available solutions, and their instantiation for the given set of requirements. The core ontology is defined in such a manner that it provides a number of fundamental concepts that have three aspects: required characteristics, available characteristics, and actual characteristics (see Figure 3.12). The fundamental concepts are defined and classified in a central taxonomy.

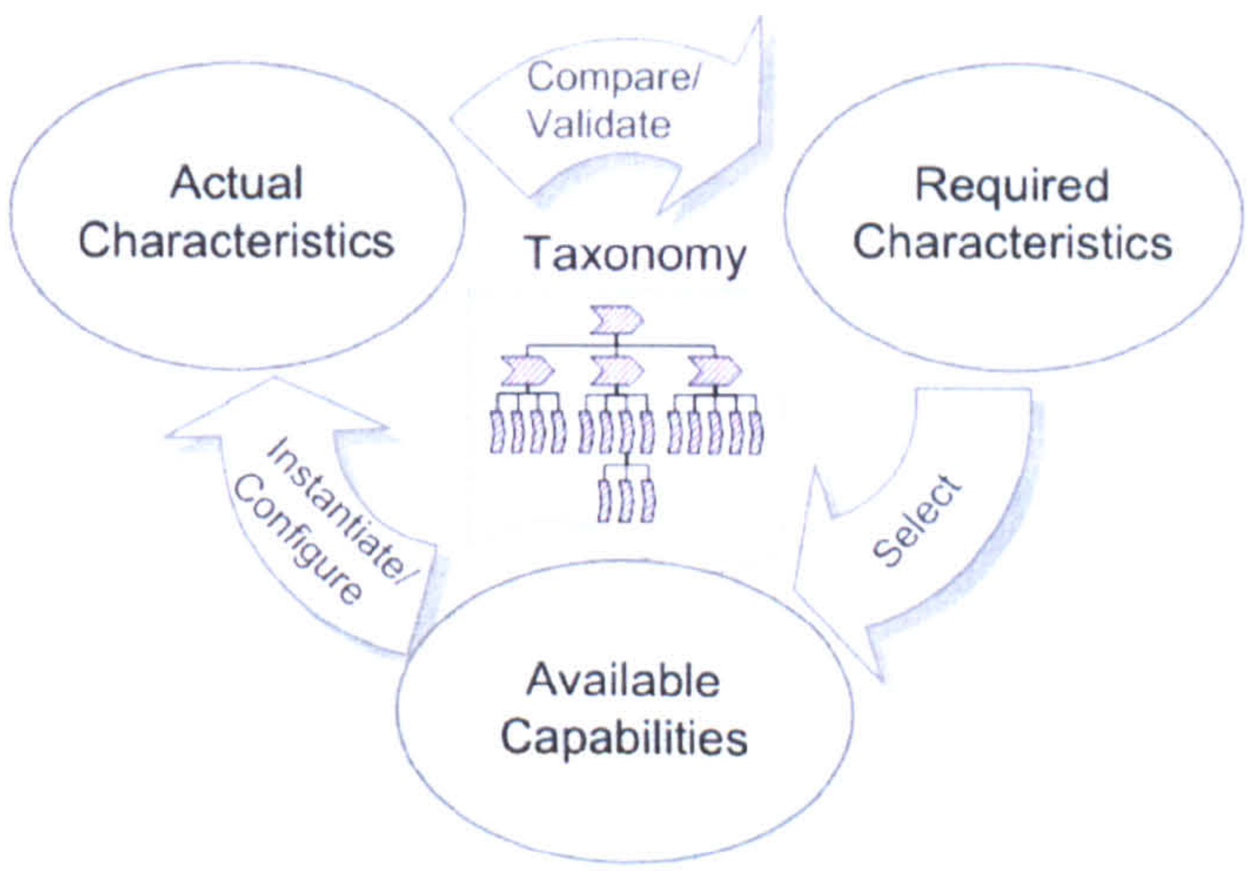


Figure 3.12 Fundamental Roles of the Fundamental Design Concepts

These fundamental aspects of the knowledge transformation during the engineering design process have been translated into the basic conceptualisation shown in Figure 3.13. The *Requirements* concept is used to define the required characteristic for a *Type* of design entity belonging to a specific *CLASS*. The characteristics for the available entities belonging to the required *CLASS* are defined through the *Type* concept. The *Occurrence* concept specifies the instantiation of *Types* for a given set of *Requirements*. The three different aspect models each have their own aggregation formalism to cover their specific needs. The requirements are defined as an Aand-Or-

Graph. The *Requirements* concept indicates AND notes and the *Variant* concept indicates OR notes in the aggregation graph. This aggregation structure has been chosen since it is possible that one or more set of requirements could be found. In this case only the more likely candidate specifications should be explored in more detail to reduce the computational effort. This formalism makes it easier to use heuristic search algorithms.

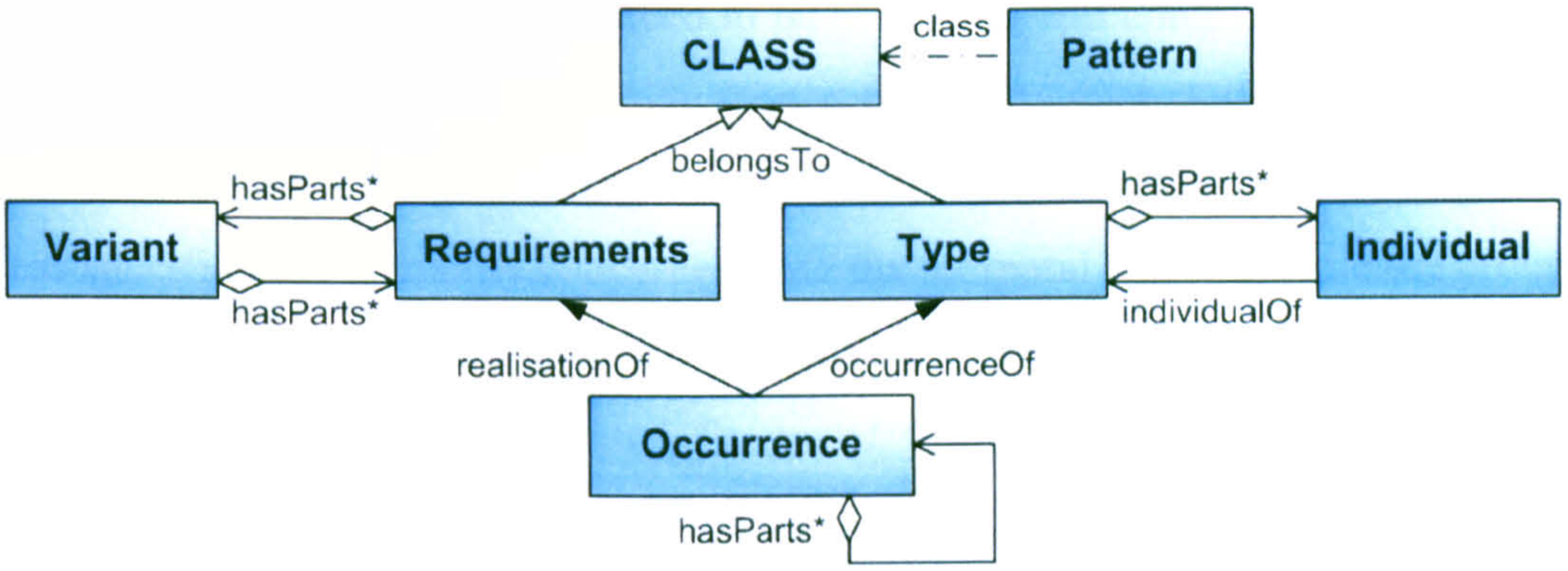


Figure 3.13 Fundamental aspect definition in ONTOMAS

The aggregation of the available characteristics is using a tree structure which is using the *Individual* concept to specify which lower level entities are contained. The Individual concept is just a placeholder that allows multiple instances of the same sub-types to be uniquely referenced. The actual specification is directly using the <hasParts> attribute of the *Occurrence* concept to define tree hierarchy. The *Type*, *Individual*, and *Occurrence* concepts are based on IEC 61346-1 [57].

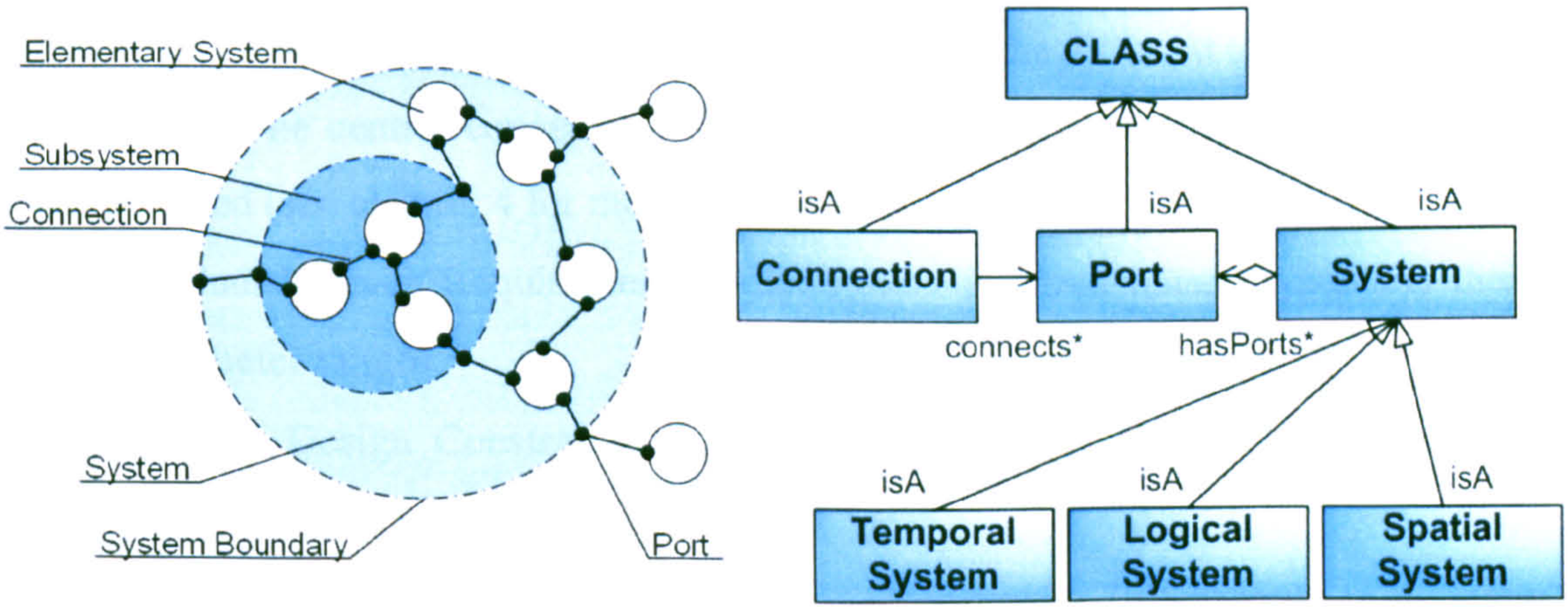


Figure 3.14 Fundamental concepts of the core ontology

Each aspect definition is linked either explicitly or implicitly to the *CLASS* concept which is the root concept for the ONTOMAS taxonomy definition. The *CLASS* concept is specialised to define the core concepts of the different domain models required for the modular assembly system design process. The domain concepts are furthermore

all defined on system theory principles using the *System* and *Port* concept to describe the different types of entities required during the design process (see Figure 3.14). The *Connection* concept specifies the topological relationships between the *Ports* of *Systems* in one domain. For example, how components in an assembly are connected to each other.

The design of assembly systems has a 4 dimensional solution space. The process capabilities of spatial entities (3D) need to be combined and controlled such that they can facilitate the assembly of a product in a timely manner. Different aspects are important at different stages during the assembly system design. For example, the process specification is mainly concerned with the temporal domain while the product and equipment definition is mainly concerned with the spatial domain. The System concept has been classified into *Temporal*, *Logical*, and *SpatialSystems* to accommodate these differences.

The more specific aspects of the different stages during the engineering design process are discussed in more detail in the following sections.

3.5.3 Product Domain Ontology Scope

The product and project domain ontology is primarily focused on capturing the user requirements of an assembly system. The product definition stands at the core of the user requirements. The required relationships, so called liaisons, between the component parts are of specific interest for the definition of an assembly system. The basic product domain ontology defines the concepts for the different aspect models on a meta-level. The central concept for the definition is the liaison concept which has been classified (see chapter 4 for more detail).

- Product Domain Requirements Definitions – required liaison types and their parameter ranges
- Product Design Constraints – liaison constraints derived from the existing solution space
- Conceptual Product Definition – Product structure (hierarchy), component characteristics, required topological characteristics (liaisons), process constraints, equipment constraints, performance requirements, and project constraints
- Product Embodiment Definition – achieved topological characteristics (liaisons), link to achieving process and system

3.5.4 Assembly Process Domain Ontology Scope

The assembly process domain ontology defines the core of the assembly system requirements. The process model is used to define the temporal order in which the individual components of the product can be put together through activities and their temporal relationships. Three distinct hierarchical levels have been defined for the assembly process definition to enable a more effective interpretation of the model. They are defined as: tasks, operations, and actions. The process domain ontology also provides a process type classification. For more detail see chapter 5.

- Domain Specific Process Definitions – required process types; required process parameter ranges; and required process structures (topologies)
- Process Definition Constraints – process constraints for their available types and parameter ranges; and realisable process decomposition patterns
- Conceptual Assembly Process Definition – required process types; required process parameters; and required process order (temporal constraints)
- Assembly Process Embodiment Definition – achieved process parameters and used process orders

3.5.5 Assembly System Domain Ontology Scope

The assembly system domain ontology defines the resource related concepts that are used or can be used to facilitate assembly processes. The central concept is the equipment which can be connected to form system solutions. Fixed hierarchical levels have been defined and incorporated into an equipment classification on the meta-level.

- Domain Specific System Architecture – required equipment types and their required process capabilities; and required equipment structures (architecture)
- System Design Constraints – equipment constraints for their available types and parameter ranges; and realisable equipment configurations
- Conceptual Assembly System Definition – required equipment types and their required process capabilities; suitable equipment structures
- Assembly System Embodiment Definition – selected equipment entities; set process characteristics; actual equipment structure

3.5.6 Equipment Definition Scope

The equipment definition is a specific subset of the assembly system domain ontology focusing on the specification of the actual equipment modules. The function-behaviour-structure paradigm has been chosen to allow a seamless transition from the functional requirements domain to the actual embodiment of the equipment (Umeda, et al. [129] and Rosenman and Gero [107]).

- **Functions** express the capabilities of a module based on the intention of the designer and are therefore subjective and domain specific. Functions are generally defined as an abstraction of behaviour for a specific use or purpose (Umeda, et al. [129]). For example the intended function of a robot is to move end effectors.
- **Behaviour** defines how equipment modules react to changes in their environment and in turn how their reactions influence the environment. For example the high level behaviour of a robot is the transformation of electrical energy into kinetic energy under the guidance of control signals.
- **Structure** defines the physical aspects of the equipment model with geometric objects, connections, and connection constraints. In the case of the robot, that would include the links and joint definitions of its structure. The attributes of the three aspect models are all based on a fully parametric model.

3.6 Summary

This chapter has given a detailed outline of the research approach behind this thesis. The approach has been systematically structured to achieve the research objectives derived from the knowledge gaps identified in chapter 2.

The vision behind this work is a holistic domain theory for the design of modular assembly systems. The requirements for the design and adaptation of modular assembly systems have been analysed. A new integrated decision making framework for the design of modular assembly systems has been proposed. The design framework shows the different development aspects of modular assembly systems and builds the bridge to understand the knowledge requirements of the domain.

The objectives of the thesis were to define a new assembly process knowledge domain ontology, develop a new assembly equipment domain ontology, and to

formulate a new method for the integrated process specification and assembly workstation configuration using the developed process and equipment models.

The fundamental hypotheses which were made at the outset of this work are that it should be possible to define a fundamental set of activities, have fixed relationships between the hierarchies of the different domains, use patterns to facilitate the decomposition and mapping between domain concepts, and to use the function-behaviour-structure paradigm to take advantage of the modular system paradigm.

The hypotheses have been turned into a new holistic ontology framework for the design of modular assembly systems (ONTOMAS). The ontology framework has a modular structure to maximise its maintainability and ease of use. The fundamental formalisms of the proposed ontology are based on system theory principles, aggregation, topological relations, and classification of fundamental concepts. These formalisms reflect the basic mechanisms of the engineering design process.

More specific definitions of the proposed ontology framework and its application in the design of modular assembly systems are discussed in further detail in the following chapters. The specification and discussion of the detailed definitions follow the principle steps of the design process and start with the proposed product model (chapter 4). The application of the ontology framework concludes the discussion in chapter 8.

4 Product Domain Ontology

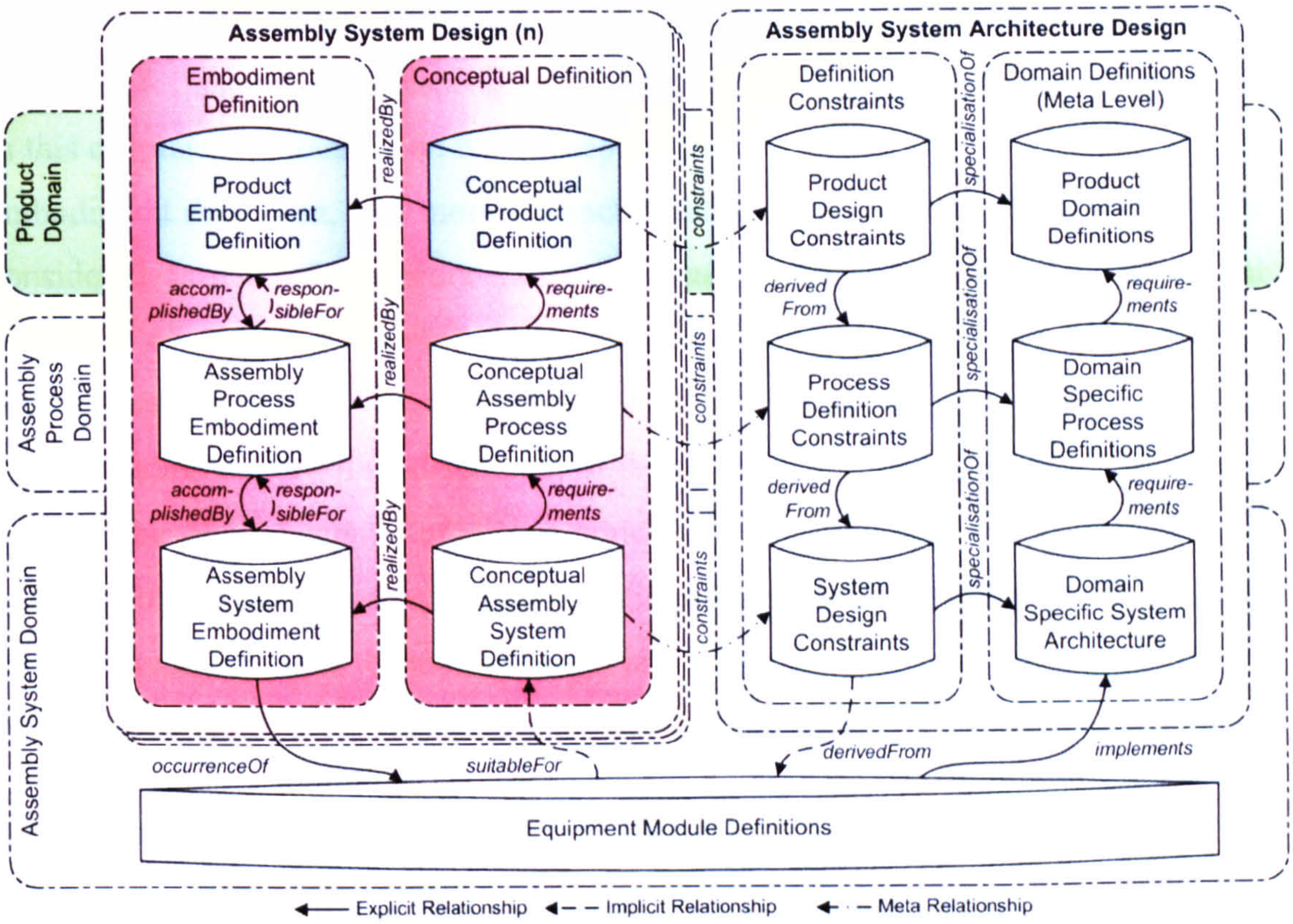


Figure 4.1 ONTOMAS Product Domain Ontology Overview

4.1 Introduction

Assembly processes define the order in which objects in the form of parts and components are assembled into a product. It is therefore important to define the link between the product model and the assembly process model to understand fully the assembly process definition. The product model defines the spatial relationships between the different objects that constitute it. The assembly process model defines the temporal order in which the objects are being assembled to form the spatial structure. That means that the assembly process is adding the fourth dimension – time – to the definition of the product.

The primary focus of this work is on the definition of the assembly relevant aspects of the product model since the main focus of this work is on the definition of assembly systems. These aspects are reflected in component related attributes and in connection (liaison) related attributes. For the definition of the assembly process, the

relationships between the product components are of the highest relevance. The objective of the model proposed here is to provide all the necessary data for a successful assembly system design. Another objective is to use the same fundamental modelling formalisms that are being proposed as foundation for the ONTOMAS ontology framework (see chapter 3 for more details).

Figure 4.1 shows which aspects of the ONTOMAS framework are being addressed in this chapter. The focus is on the conceptual product definition and its eventual final embodiment definition. The more abstract constraints and domain definitions are not considered as part of this work since the focus is not on the design of the actual product. Consequently, the model only needs to be able to describe the results of the product design.

The chapter starts with a discussion of the informal terminology definitions chosen to describe the concepts of the product domain ontology. This leads to the examination of their representational requirements. The next section describes how the requirements have been translated into a product domain conceptualisation. Finally, the chapter concludes with a summary of the proposed domain model.

4.2 Informal Product Domain Description

The product domain model describes the characteristics of the objects which need to be assembled and the characteristics of their relationships. In this section the fundamental terminology within the product domain will be informally described to convey a fundamental understanding of the concepts of this domain before a more detailed definition of the modelling approach for the product structure is shown.

The product, the central term that gives this domain its name, is defined by the Oxford English Dictionary as:

"That which is produced by any action, operation, or works; a production; the result. Now freq. that which is produced commercially for sale." OED [90]

The following definition will be used throughout this work.

Definition: Product

"A product is the result of a manufacturing process with the intention to sell it."

A product can either be a single piece part or an assembly of piece parts. The focus of this work is on the assembly aspect of the manufacturing process. The products

considered in this work are therefore assemblies in the majority of cases. The assembly is, for the purpose of this work, the most fundamental concept within the product domain. The Oxford English Dictionary defines an assembly in the most general sense as:

"A collection of things" OED [90]

Homem de Mello and Sanderson [55] define mechanical assemblies more specifically as:

"... a composition of interconnected parts forming a stable unit."

This definition includes the notion of connection between the entities that belong to an assembly. This excludes for example a box full of parts from the definition of an assembly even though they are also a collection of things. These types of collections are in manufacturing more commonly called batches. Furthermore Homem de Mello and Sanderson's definition includes the concept of stability which requires the entities of an assembly not only to be connected to each other but to be connected in a manner that the resulting collection of entities is stable in its own right, once complete. The following definitions for assemblies and batches will be used throughout this work:

Definition: Assembly

"An assembly is a stable collection of interconnected objects."

Definition: Batch

"A batch is a collection of one or more identical objects."

The objects that make up an assembly are traditionally called parts and components. Parts are generally defined by the Oxford English Dictionary as:

"A piece or section of something which together with another or others makes up the whole." OED [90]

A more restricted definition will be used for parts throughout this work since this work is specifically concerned with the definition of assemblies.

Definition: Part

"Parts are the smallest physical objects an assembly is made of."

The definition of a part is closely linked with the definition of the physical object it represents. In fact, the only difference that has been attributed to the part definition is that it defines the manufacturing process of the physical object. This means that the physical object that is the input to the assembly process is only the final stage of the part definition process.

The concept of components is commonly used to represent assemblies or parts which are the input of the assembly process which is being considered. Their precise definition might not be known or is not important for the definition of the assembly process and can subsequently be abstracted to a sufficient level of detail. For example it is in most cases not important to know the detailed history of the parts before the assembly process.

As a result, components represent the smallest entities of the assembly from the viewpoint of the stakeholder who owns the assembly process. From this viewpoint the only essential difference between components and parts is that components could have functions of their own, including internal degrees of freedom, since components could be representative definitions of assemblies. For example, an electric motor that is used inside an electric shaver will be treated as a component by the manufacturer of the shaver. The same electric motor, however, would be an assembly for the maker of the motor.

For the above reasons the definition of an assembly will be based on components as the smallest entities throughout the rest of this work. Hence components are defined as:

Definition: Component

"Components are all the entities that are supplied to the assembly process to make up the assembly. Components are abstracted representations of parts and assemblies."

It is important for the specification of the assembly process to understand the characteristics of physical connections between the components in an assembly. The term assembly liaison has been established by Bourjault [13] (see Henrioud and Bourjault [49]) for the definition of these connections. The Oxford English Dictionary defines liaisons as:

"An intimate relation or connexion" OED [90]

A more restricted definition will be used in this work to define liaison within the assembly domain:

> **Definition: Liaison**
>
> "A liaison is the physical connection between two components in an assembly."

4.3 Domain Model Requirements

The requirements for the product domain model are discussed in this section. Since the focus of this work is primarily on the assembly process and resource specification, the product model only needs to capture those attributes that are required to guide the decision making process during their specification. The model should however be defined in such way that it caters for a seamless extension of the product domain model to include other details which are not relevant to this work. Furthermore, the chosen modelling paradigm needs to be suitable for the exchange of information between different stakeholders over different media including the internet since a distributed iteration process has been proposed for the concurrent decision-making during the assembly workstation design.

At this point it needs to be clear that the aim of this product model cannot be the complete definition of all the attributes of a product with all its parts but rather a more goal-specific definition of just those characteristics of the product that are important for the definition of assembly processes and assembly systems.

All the characteristics of a product and other information relevant for the assembly process will be analysed in this chapter. Based on their importance for the definition-process of assembly systems they will be either directly included or abstracted to fulfil the needs of later decision making-processes.

4.3.1 Components

Components are the smallest entities a product will be broken down into for a specific assembly process. For example, components could be nuts and bolts but also more complex assemblies like motors or bearings. In the wider sense less defined entities like grease and glue can also be considered as components. Components are only fully defined with all their physical characteristics. These include their material

properties, geometry and information about previous manufacturing processes and their influence on the properties of the component. For more complex components that are assemblies in their own right this includes also a definition of their internal degrees of freedom and intended functions.

The relation of the component to the functions of the product is also significant since it provides an indication of what kinds of tests would need to be performed to insure the correct function of the assembled product. This however is outside the scope of this work.

Geometry:

Three main purposes for the definition of the component geometry can be identified. Firstly, it allows computer-based reasoning about the behaviour of components. Secondly, it provides the infrastructure to attach additional information to the component. This could for example include tolerances and contact conditions. Lastly, it provides the basis for the visualisation of the components.

For this work it is enough to represent the components as abstract entities that do not have any explicit connection to their actual physical topology. This means that the model does not provide information about the 3D shape of the individual component in a manner that can be used as basis for any computer based-reasoning. The model is, however, built in such a way that it can be extended to include this information.

At this stage the geometrical information of the components only needs to be available for visualisation. Two types of visualisation should be provided: 2D for illustrative purposes and 3D for more detailed visual impressions. Adequate 2D formats include JPEG, GIF, BMP, etc. 3D models can be provided using either standard CAD exchange formats like IGES or STEP or by using formats that are more suitable for web applications like VRML.

The aspects that need to be directly derived from the geometry for the specification of the assembly equipment include:

- The **handle-ability** of the component defining how easy it is to hold and move the component during the assembly process. Additionally, there should also be a formalism that allows specifying the results of the handle-ability analysis in terms of holding points.
- The **orient-ability** of the component to give an indication of how easy it is to orientate the component.

Material Properties:

The material properties of a component are depending on the chemical composition of the material as well as how the material has been treated. The material properties might vary throughout the components depending on how they have been manufactured or if they are assemblies in their own right.

Careful consideration has to be given to the right level of decomposition of the material properties. Only if the breakdown has a significant impact on the assembly of the component should it be included.

Some of the aspects which need to be available for the assembly process and system specification are derivatives of the geometry and material properties of the component. These include:

- The **fragility** of a component to give an indication of the allowable forces that can be applied to it. Some components could have different fragility values depending on the geometry of the components and the direction of the force. In this model, however, we will adopt a unified fragility for each component. This value should be chosen with the assembly operations in mind that are likely to be applied to the component.
- The **sensitivity** of a component expresses how sensitive the component is during the assembly process to surface damages like scratching. This value should reflect the resistance of the component against scratching.
- The **visibility** of the component defines how important the undamaged appearance of the component is for the overall appearance of the product. This value, together with the sensitivity of the component, gives an indication of how much importance has to be given to the protection of the component surfaces during the assembly process.
- The **weight** of the component is important for the definition of the right handling and tooling equipment.
- The **flexibility** of the component defines how easy the component deforms. This attribute however is very subjective and should be based on how critical this is for the assembly process.

Some of the attributes need to be defined from the view of the assembly process and system specification. This might be critical if the values have to be defined by the

product designer who does not necessarily have adequate knowledge of assembly processes.

Process History:

The process history of a component gives an indication of what has happened to the component before it is being used in the assembly process. For example, previous machining processes and ways in which the component could be supplied to the assembly process give an indication of how the component can be fed to the assembly system.

4.3.2 Product Hierarchy

Products are normally structured by grouping components into subassemblies. Subassemblies are defined for different reasons and with different motives. Some reasons for example include creating modules that can be used in a wide variety of products, to reduce the manufacturing cost, to separate different stages in the assembly process, etc.

Because there are so many different motivations for the specification of subassemblies, the grouping of components into subassemblies cannot easily be automated using computational intelligence. The model needs, however, to provide the means to structure the product.

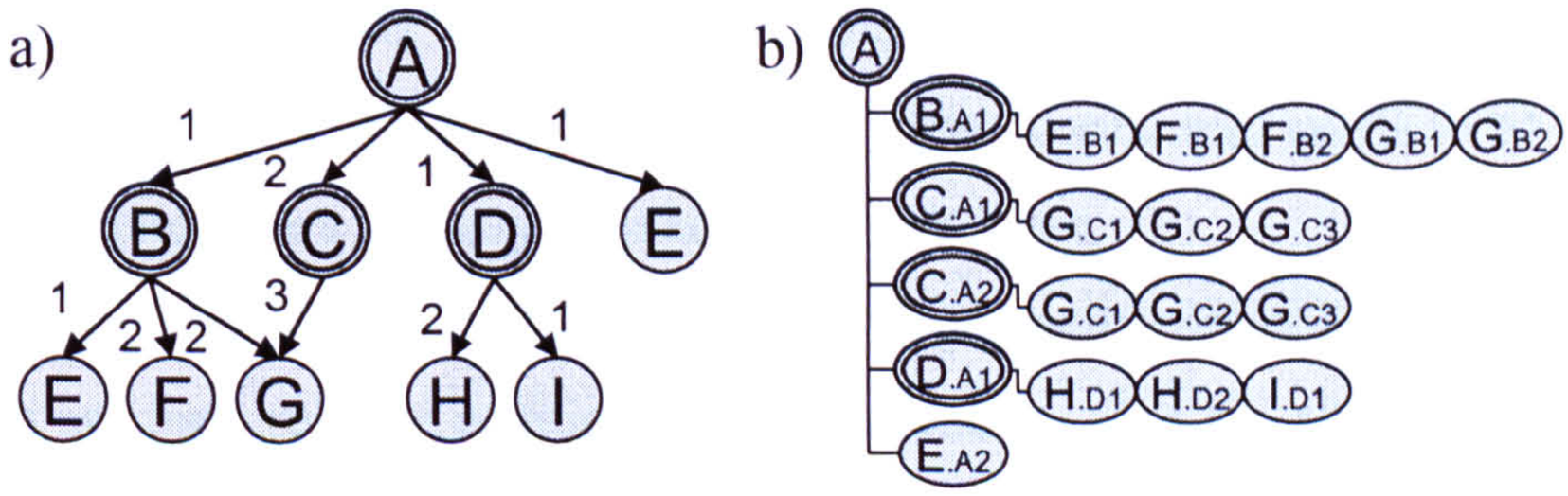


Figure 4.2 Example product structure represented as a) graph and b) tree structure

Figure 4.2 shows an example of a product structure, where a) shows the graph representation where the connection defines the number of components required and b) the tree representation. A is the product to be assembled, B, C and D are subassemblies in the product structure and E, F, G, H and I are the components that are used to assemble the product. The definiton of the product hierarchy needs to be able to be translated into either of the above representations.

4.3.3 Product Structure

The assembly process is essentially the temporally ordered establishment of the component liaisons. The type of liaison between components prescribes the required assembly process. The parameters of the required assembly process are derived from the characteristics of the related components and of the liaisons themselves.

On the most basic level liaisons should only represent the physical relationship between two components. These kinds of liaisons define for example how two components fit together if they have contacts. Figure 4.3 shows an example of relations between components where numbers represent the relations and capital leters represent components.

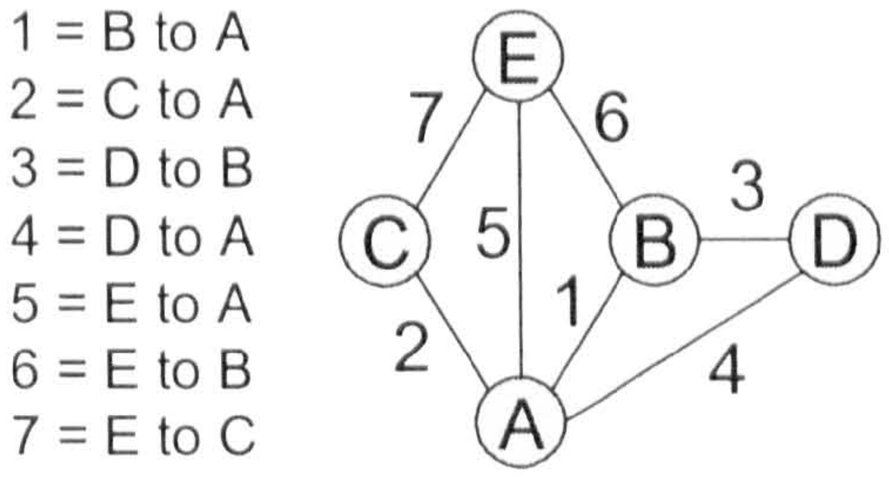


Figure 4.3 Example of relationships between components.

Two aspects need to be defined to model the basic relationship between two components: their geometric relationship in terms of their relative distance and orientation and the manner in which they are related to each other in terms of connection mechanism. The relative position of components only needs to be defined when the components are actually connected to each other.

As mentioned above, the driving factors for the geometric relationship between the components are their relative positions and orientations. Also the allowable variation from the specified values has to be defined. This is particularly important for the definition of the required precision during the selection of the right assembly equipment.

More complex connections require not only the modelling of the geometric relationships but also how the connection is being established including process parameters and material or energy requirements. The material or energy needed to define the connection will be called attachments based on the definition by Henrioud and Bourjault [49] and Homem de Mello and Sanderson [54].

The model therefore has to consider two types of relationships: basic relations between parts e.g.: a bearing fit on a shaft (see Figure 4.4 a) and relations that are

using a secondary component to establish the connection between two components: e.g. cover is fixed on base using screws or rivets (see Figure 4.4 b).

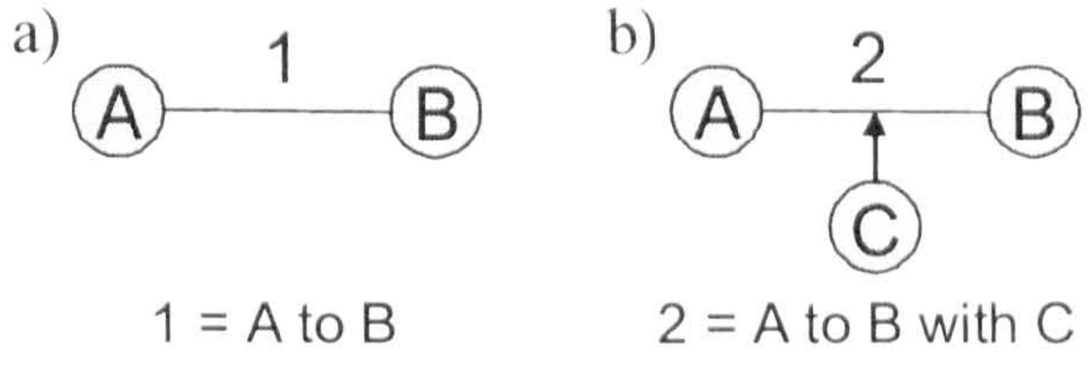


Figure 4.4 Basic types of liaisons between components a) simple b) with attachments

Some liaisons have two states in cases when a attachement element needs to be transformed during the assembly proces. For example, a rivet has one state when it is being inserted and is then deformed to establish the fixed joint between the two components.

For each assembly in the model it is important for the grouping of assembly processes to understand if the assembly is stable and under which conditions. This information needs to be deduced from the sum of the liaisons that connect all the components in the assembly.

4.4 Formal Product Domain Concept Model

The product domain concepts discussed above are defined formally in this section. The modelling mechanism is based on ontology engineering principles discussed in chapter 3. The product domain conceptualisation extends the core ontology discussed in the same chapter.

Figure 4.5 shows an overview of how the informal definitions discussed above translate into a more formal concept structure. A *Product* concept can be either be an *Assembly* or a *Component* depending on its place in the product hierarchy. *Assemblies* are defined as sets of components and subassemblies that are connected with *Liaisons*. A *Batch* is a collection of *Components* or *Assemblies*, but unlike the *Assembly* there are no defined *Liaisons* between the constituents in a *Batch*. The definition of a *Component* is normally directly associated to an *Object* that represents its physical characteristics to the required level of detail. More complex *Components* that represent *Assemblies* could have a degree of freedom. To model this aspect it might be required to represent the physical characteristics of the *Component* with more than one *Object* and define their *Connection*. The *Connections* between

Components inside an assembly are defined by *Liaisons*. They are part of an *Assembly* definition if they connect two *Components* that both belong to the same *Assembly*.

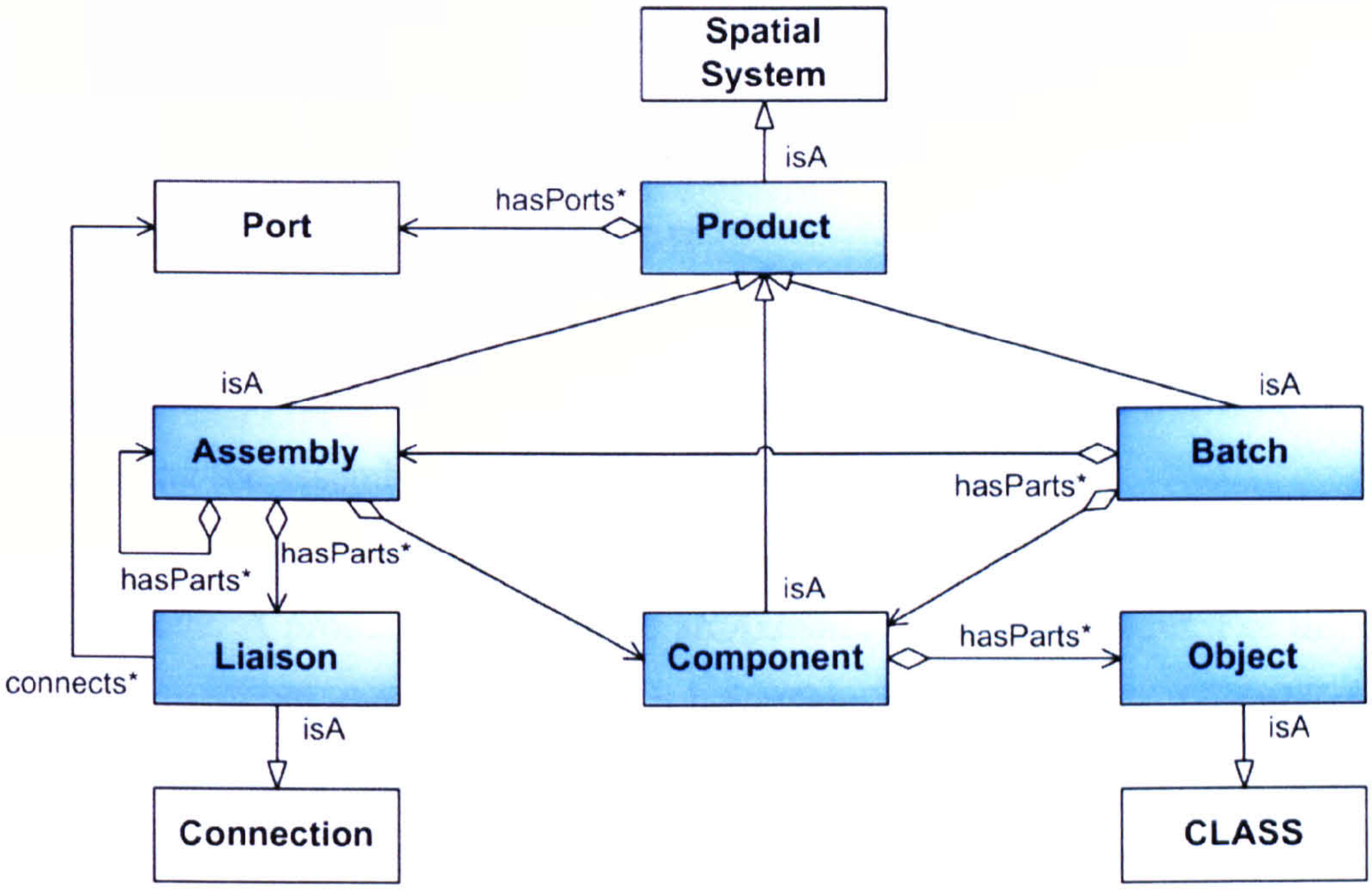


Figure 4.5 Product Domain Concept Classification and Hierarchy Overview

All concepts of the product domain ontology are either directly or indirectly derived from the central *CLASS* concept. Consequently, they all inherit the three aspect definitions described in chapter 3. The aspect address the specification of the requirements for a specific concept class, available characteristics for the concept class, and the actual characteristics that an instance of the class has been set to fulfil the requirements.

The *Liaison* is the central concept of the product domain ontology since it describes the connection characteristics the assembly system will have to achieve to assemble the product successfully. As a result, only the *Liaison* concept will be considered from the requirements point of view. All the other concepts are assumed to be fixed.

The requirements aspect of the *Liaison* concept defines how the product designer intends for the *Component* of the *Product* to be connected to each other. This definition provides the central input for the assembly system design. The available *Liaison* characteristics are derived from the available assembly process characteristics which will ultimately result from the chosen equipment configuration (see chapter 5 and chapter 6 respectively). The actual *Liaison* characteristics are similarly derived from the actual assembly process characteristics once they have been defined.

The following sections give a more detailed definition of the different product domain ontology concepts. The definition of the domain concepts does not distinguish between their specification aspects since their primary difference is only in their instantiation.

4.4.1 Product Structure Definition

In this section the product domain specific concepts will be formally defined. The definition of the product domain concepts is focused on the characteristics that are required during the assembly workstation design process.

The description of the product structure is based on *Assemblies* and *Components* as discussed above. A product can either be an *Assembly* or a *Component*. For the work it has been assumed that the inverse relationship is also true – *Assemblies* and *Components* are all *Products*. Based on this assumption both the *Assembly* and *Component* concept are defined as extensions of the *Product* concept.

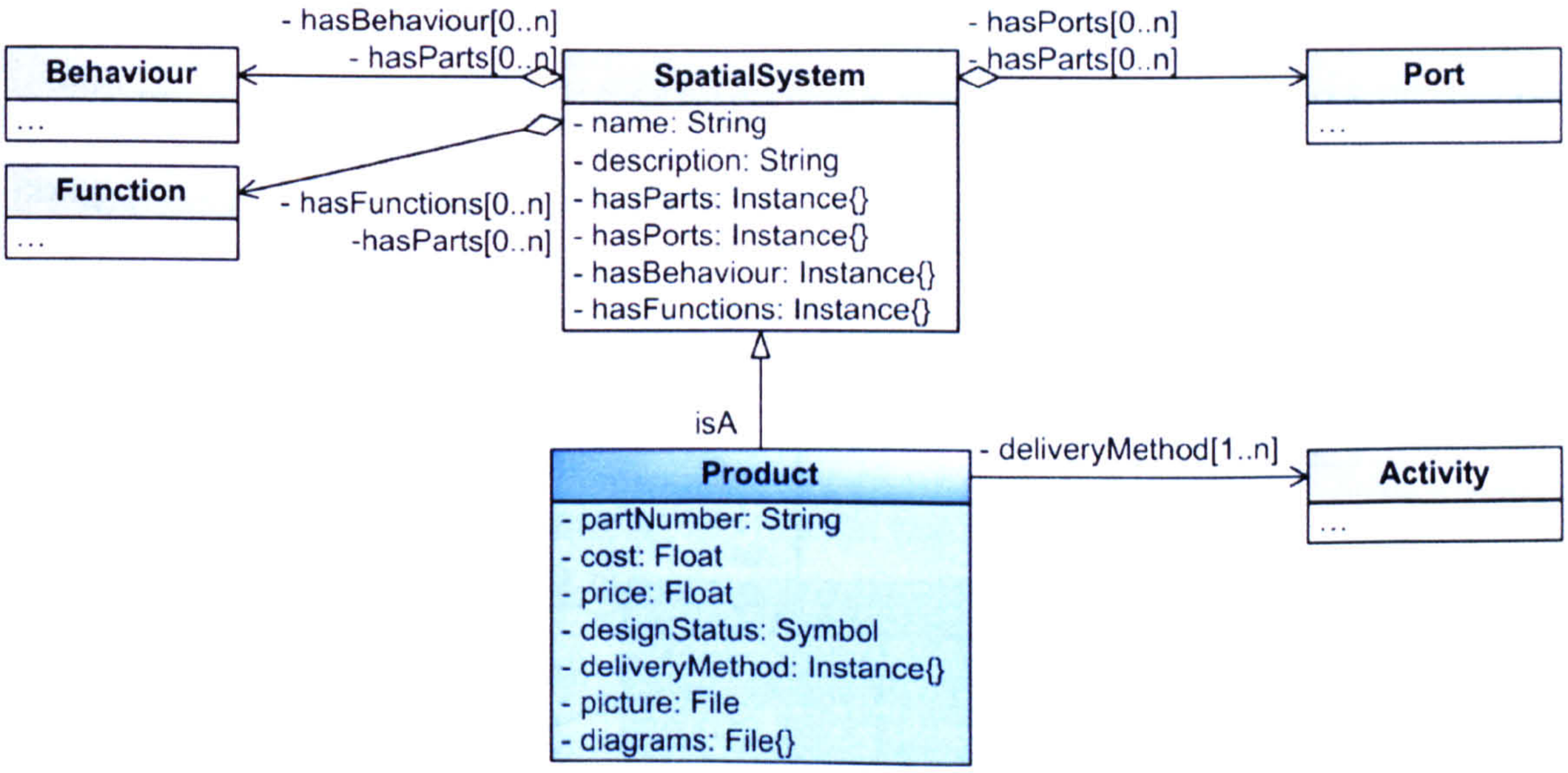


Figure 4.6 Product Concept Definition

The *Product* concept is therefore the central concept of the product domain. Following general system theory, the product is being considered as a *System*. It has been defined as one specific type of *SpatialSystem* which means that every product has a behaviour and function definition and that a *Product* interacts with its environment via defined *Ports* on its boundary. Furthermore, products have additional attributes that define how they are identified, costing information, design status, and any additional material that is relevant to the product (see Figure 4.6).

Component concepts define the lowest level of granularity required of entities for a given assembly. The *Component* concept extends the *Product* concept (see Figure 4.7). *Components* also need to define the assembly specific characteristics of the entities that are being put together (see section 4.3.1). The <fragility>, <sensitivity>, <visibility>, and <flexibility> attributes of the component are defined with a five-value ranking that needs to be specified by a domain expert. The scale used is based on natural language symbols including with increasing importance: 'none', 'small', 'medium', 'high', and 'very high'. A similar approach has been used to define the handle-ability and orient-ability value of the component. The scale used is: 'very easy', 'easy', 'neutral', 'difficult', and 'very difficult'. The chosen approach is not objective and the results will be different for different domain experts. This has been done under the premise that the above component characteristics will only be used to make more focused decisions. The geometric and material properties of the *Components* are defined by associating *Objects* to the *Component*. The object model is described in more detail below (section 4.4.2). Degrees-of-freedom that might occur in more complex *Components* are modelled by linking two *Objects* to the component and defining their relationship.

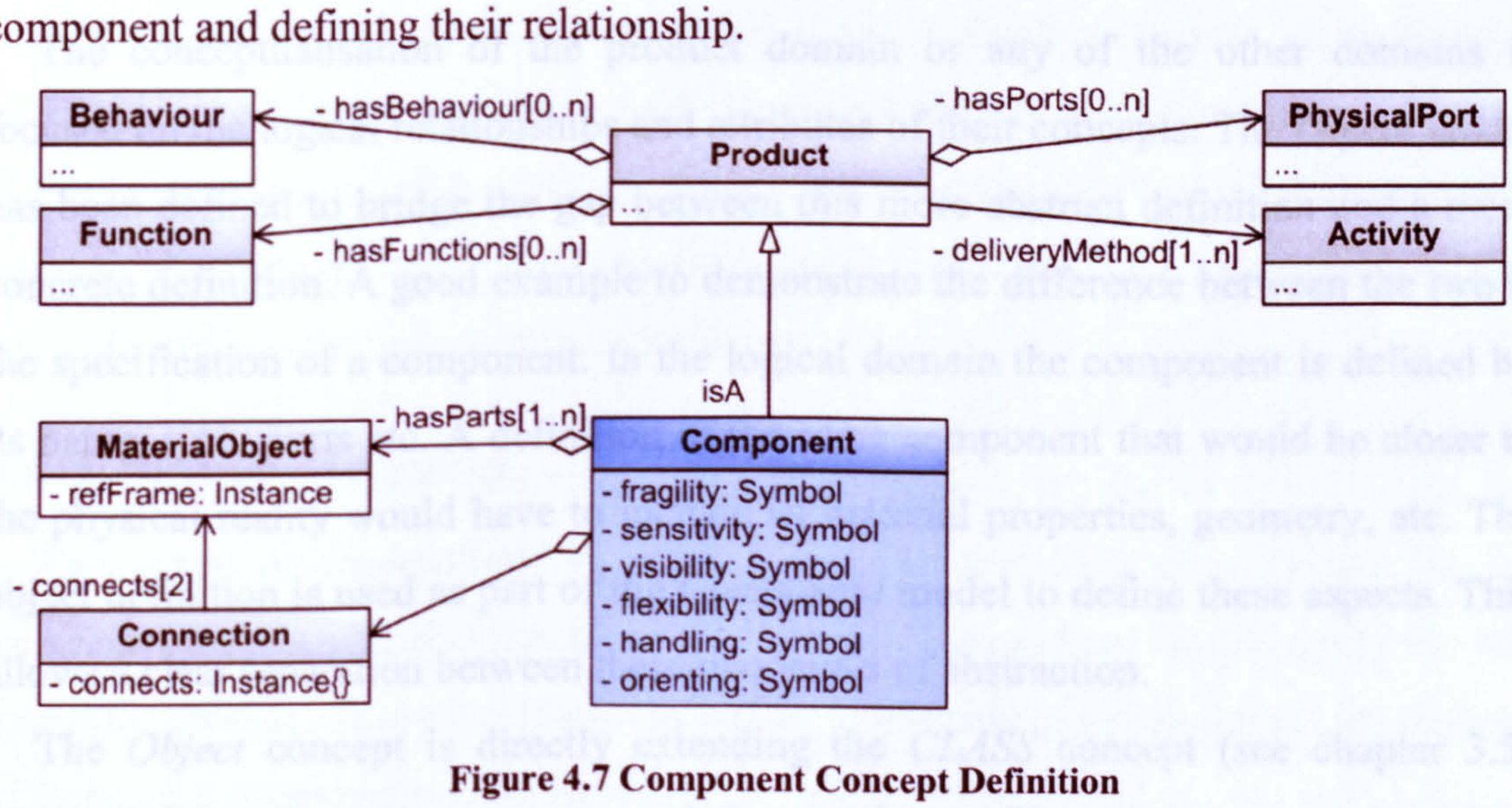


Figure 4.7 Component Concept Definition

Assemblies have been defined as a separate concept to specifically capture the needs for the assembly process. The underlying assumption behind this definition is that all separately defined assemblies have a sufficient degree of stability to transport them between assembly processes. Based on this assumption it becomes possible to establish a relationship between the definition of the assembly and an assembly cell on the equipment side to facilitate its assembly process. As discussed above an assembly is a specific type of product. It therefore extends the product concept and

inherits all the attributes from the product definition (see Figure 4.8). The <hasParts> attribute of the assembly is defining the components, subassemblies, and liaisons it comprises. Furthermore, the assembly definition needs to contain some additional parameters that are relevant to its assembly since it could be used as part of another assembly. The <hasParts> attribute is constrained to include at least two components or subassemblies in any combination and at least one liaison between them.

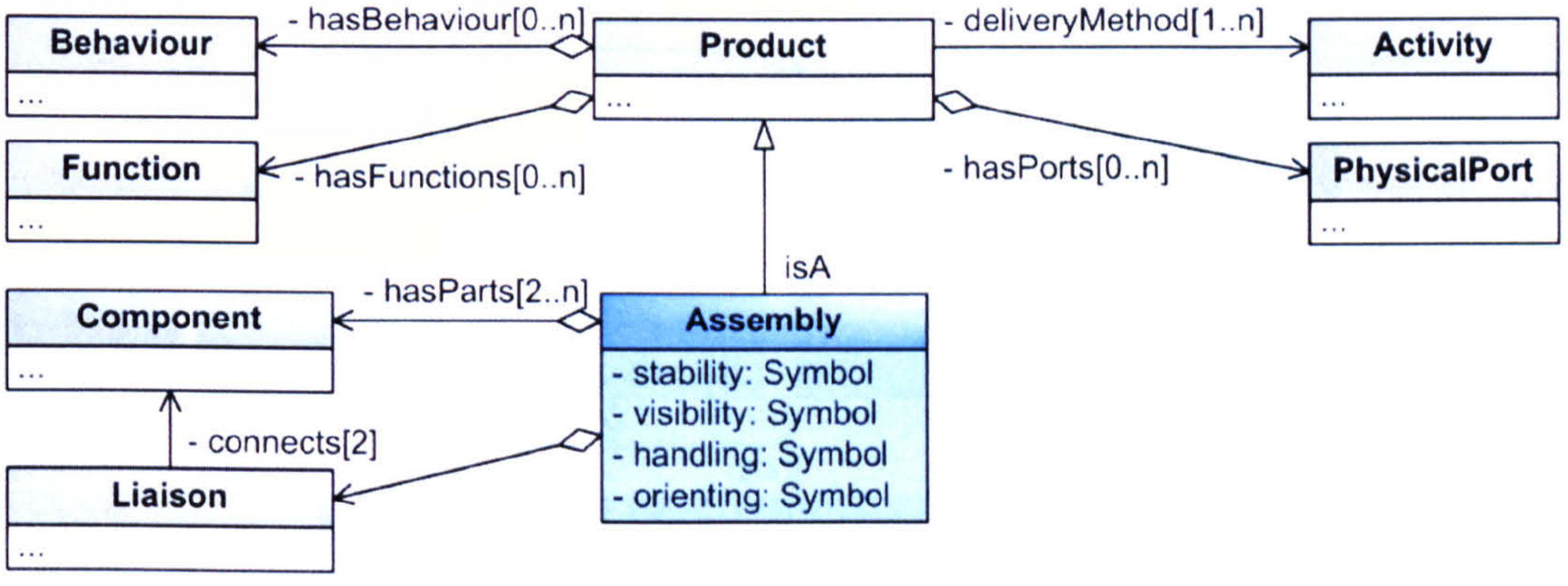


Figure 4.8 Assembly Concept Definition

4.4.2 Object Definition

The conceptualisation of the product domain or any of the other domains is focused on the logical relationships and attributes of their concepts. The *Object* model has been defined to bridge the gap between this more abstract definition and a more concrete definition. A good example to demonstrate the difference between the two is the specification of a component. In the logical domain the component is defined by its name, type, ports etc. A definition of the same component that would be closer to the physical reality would have to include its material properties, geometry, etc. The object definition is used as part of the *Component* model to define these aspects. This allows a clear separation between these two levels of abstraction.

The *Object* concept is directly extending the *CLASS* concept (see chapter 3.5) which defines the taxonomy root of the ontology. *Objects* are categorised into material, information, and energy objects (see Figure 4.9). This categorisation is based on the definition of Pahl and Beitz [94] that has also been used by Welch and Dixon [135] as foundation for their function, behaviour, and structure definition.

Welch and Dixon [135] further categorise energy objects into translational, rotational, electrical, thermal, and fluid energy. The material object domain is generally divided into solid, fluid, and gas objects. The object classification shown in

Figure 4.9 will be the basis for the definition of all the physical entities within the environment of an assembly system throughout this work.

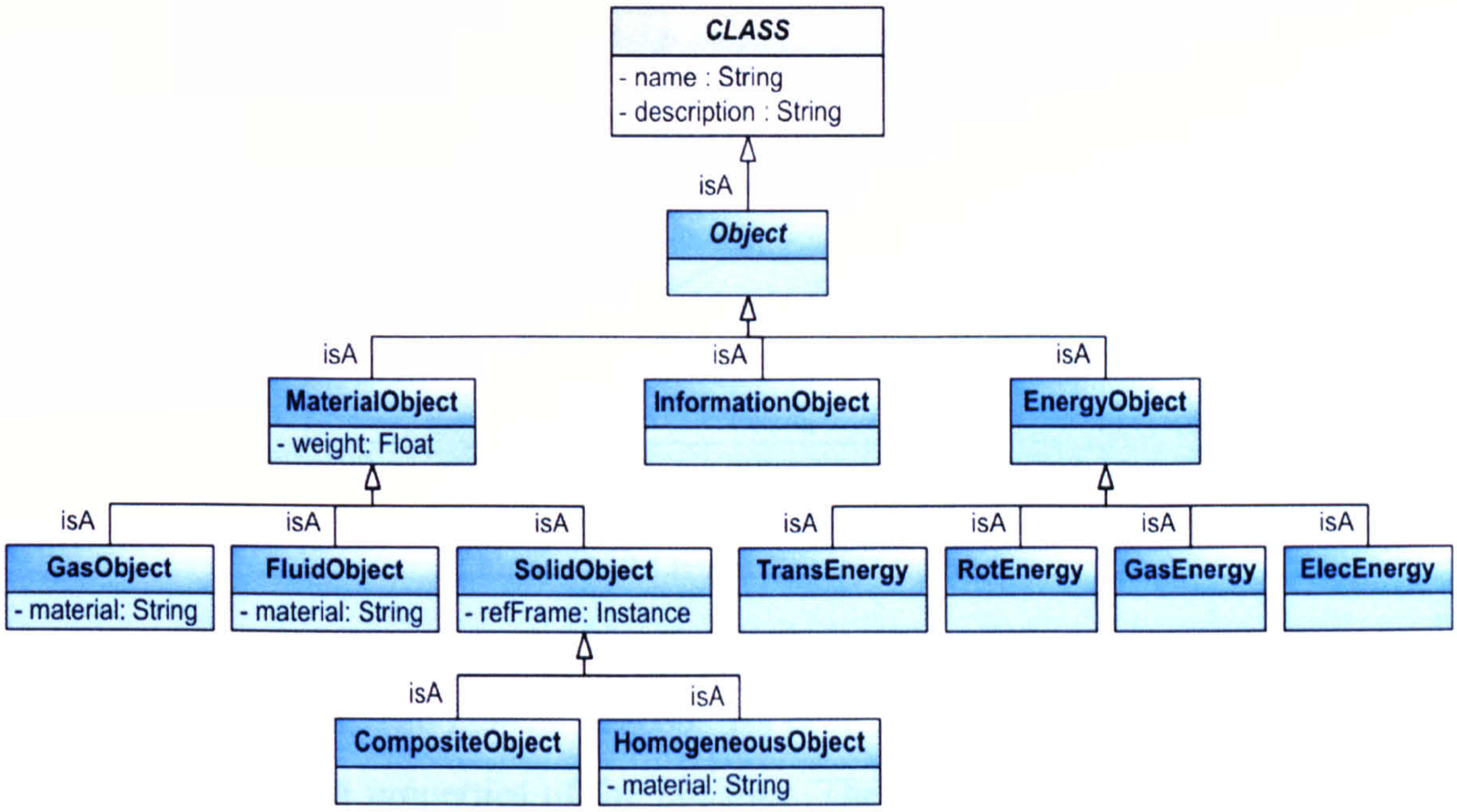


Figure 4.9 Object Concept Classification

The objects which are associated with components in an assembly are, in their majority, solid objects. There are rare cases in which fluids and gases are included in an assembly. In these cases they could also be considered as components. This work is primarily focused on the design of assembly workstations for solid objects. The chosen modelling approach does, however, not prevent the later integration of non-solid objects into the proposed methodology.

4.4.3 Product Topology

Two types of connections are relevant for the modelling of a *Product* in the assembly domain. Firstly, the *Liaisons* between *Components* which prescribe the assembly process. Secondly, the joints inside a component which define the degrees of freedom a component might have.

The *Liaison* concept defines the connection between components and/or subassemblies. The *Liaisons* are defined as specialisation of the *PortConnection* concept (see Figure 4.10). This restricts *Liaisons* to connect only *Components* with their *Ports* in accordance with system engineering principles (see chapter 3.5). The <hasParts> attribute of the *Liaison* concept has been restricted to allow only other *Connections* to be directly part of the *Liaison*. Lower level connections could for

example be the geometric relationships between features of the connected *Components*.

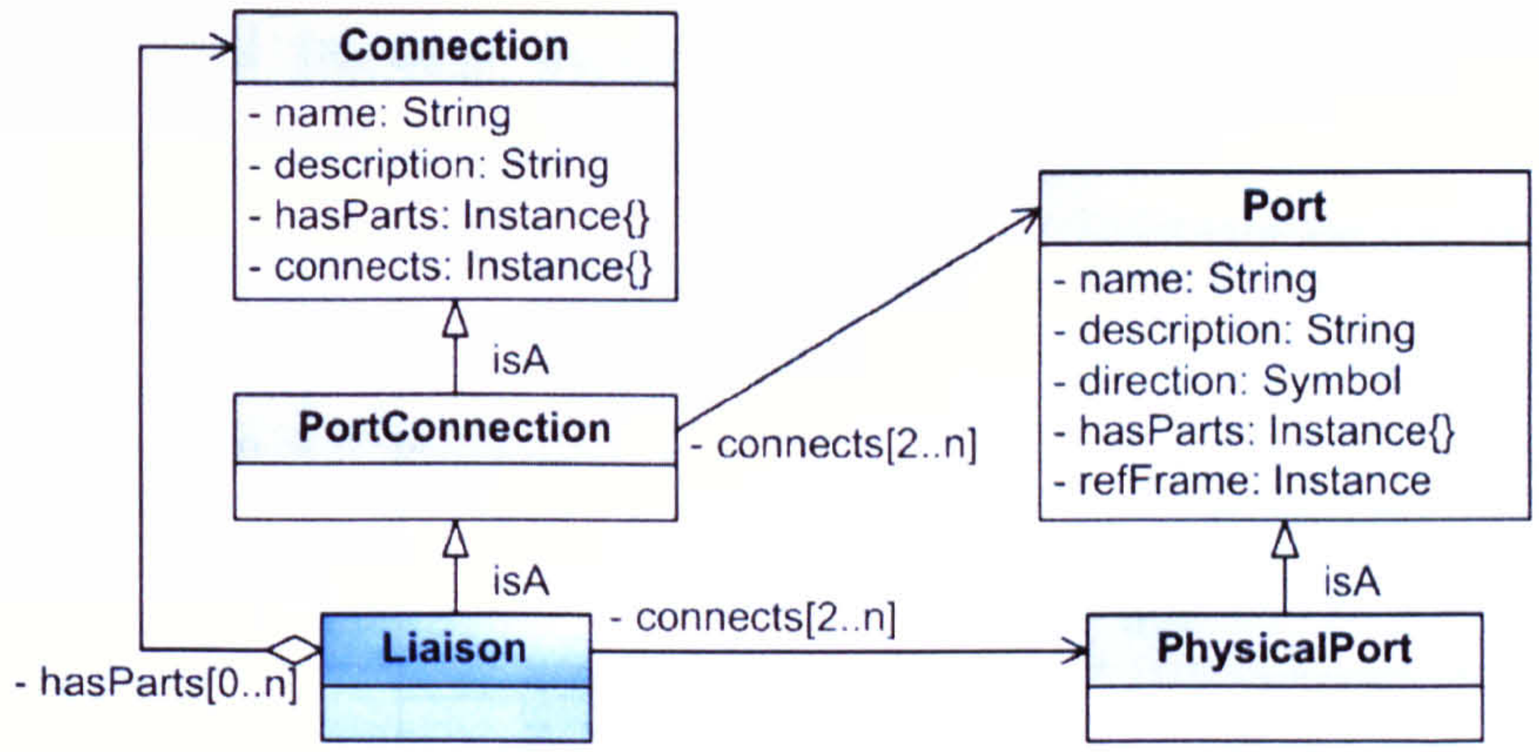


Figure 4.10 Liaison Concept Definition

The type of *Liaison* is the driving factor for the later selection of the appropriate assembly operation. The *Liaison* concept has been classified into three main types to reflect the different properties of the *Liaisons*. The main *Liaison* types are (see also Figure 4.11):

- ***ContactLiaisons*** relate points, lines, or surface to each other. They are the most basic *Liaison* between two *Components* and they have on their own very limited degree of stability.
- ***FitLiaisons*** define how two boundaries fit into each other. This type of *Liaison* is quite common and has a higher degree of stability. It can be argued that fit liaisons can be represented as combinations of *ContactLiaisons*. The most well-known example for a *FitLiaison* is the peg in hole where there is a *FitLiaison* between the outer surface of the peg and the inner surface of the hole.
- ***ConnectionLiaisons*** establish a normally permanent and often irreversible connection between two *Components*. Good examples for a *ConnectionLiaison* are welding, soldering, crimping, etc. Some of these *Liaisons* require a third component or material to establish the connection, for example rivets or solder. These *Components* can be linked to the *Liaison* definition via the <attachments> attribute. The attached *Component* also has to be part of the *Assembly* the *Liaison* belongs to. The *ScrewConnectionLiaison* is on the boundary between being a *FitLiaison* and a *ConnectionLiaison*. It is reversible but at the same time self locking and as good as permanent. The distinction has been made between components whose purpose is solely the

connection of other *Components* – classical screws – and *Components* that have a thread as part of their features to connect to other *Components*. The former would be in a *ScrewConnectionLiaison* and the latter would be connected with a *ScrewFitLiaison*.

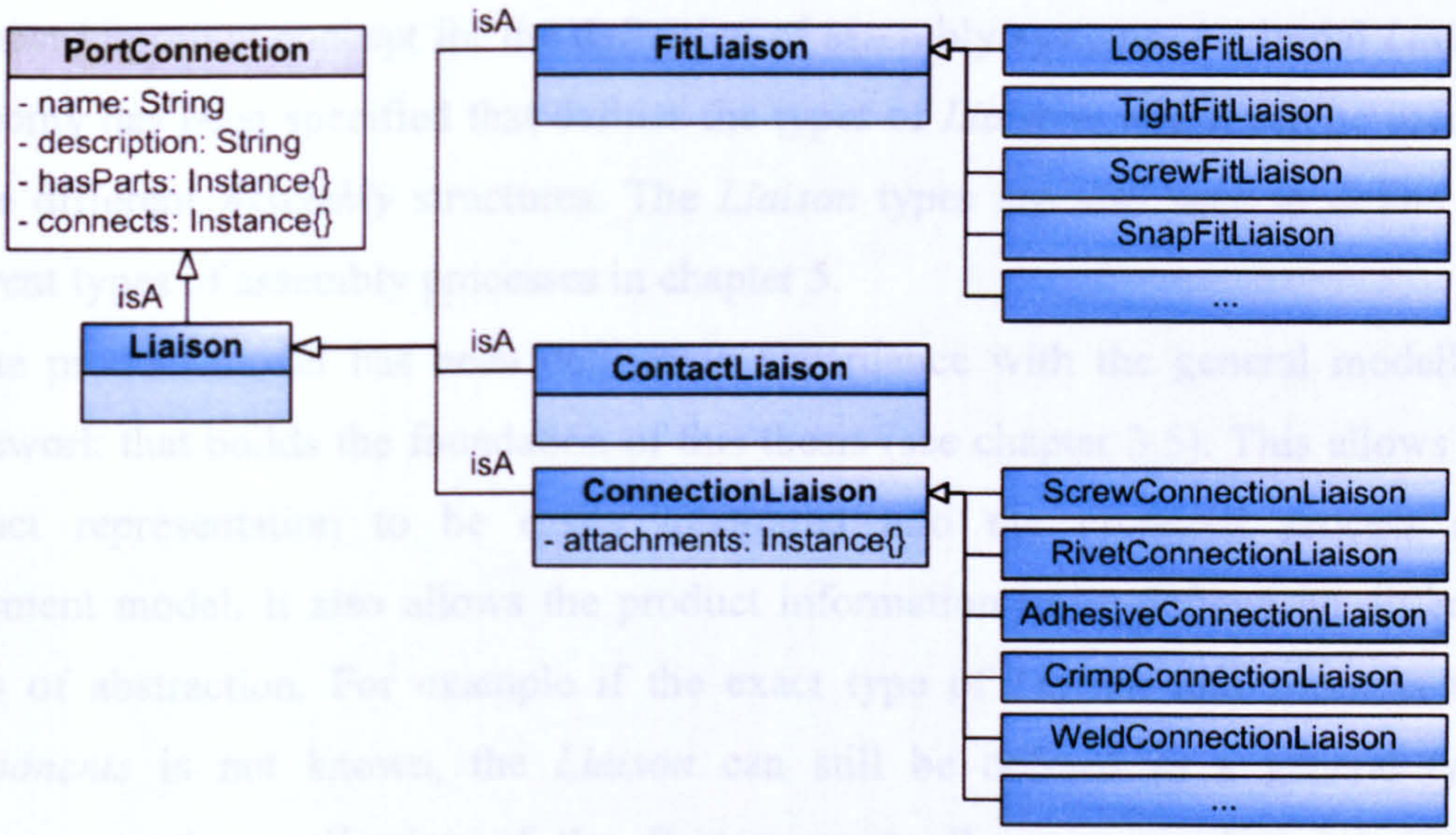


Figure 4.11 Liaison Classification

The classification of the *Liaisons* allows not only an easier selection of the right *Liaison* type but also to define more abstract *Liaison* at an early design stage to get a first impression of its implications.

The specific attributes of the different *Liaison* types are based on the geometric relationship they establish between the components. For this work it has been assumed that a domain expert will provide the necessary information where and when required. The proposed structure does however allow the information that would be required to provide a higher level of automation to be included in the model.

The *Joint* concept defines the connection between the solid objects that define the physical characteristics of a component. They essentially have to define the degrees of freedom a component might have. A bearing could for example have one rotational degree of freedom. This is important to understand the handle-ability and orient-ability of the components during the assembly process. The joint connection is even more relevant for the modelling of the kinematic characteristics of equipment. A more detailed definition of the joint connection is therefore given in chapter 6.

4.5 Summary

In this chapter the product domain conceptualisation of ONTOMAS has been defined and discussed. The central concepts of the product domain are the *Product* itself and the *Liaisons* between the *Components* in a *Product*. The *Liaison* concept is the most important concept for the definition of assembly systems. An initial *Liaison* taxonomy has been specified that defines the types of *Liaisons* which can be used to define different *Assembly* structures. The *Liaison* types are also used to define the different types of assembly processes in chapter 5.

The product model has been defined in accordance with the general modelling framework that builds the foundation of this thesis (see chapter 3.5). This allows the product representation to be easily integrated into the proposed process and equipment model. It also allows the product information to be defined on different levels of abstraction. For example if the exact type of *Liaison* that connects two *Components* is not known, the *Liaison* can still be defined as a general type. Furthermore, the application of the *Port* concept allows a seamless integration between the concepts of the product domain and the concepts of the equipment domain. This is a big advantage since the *Components* need to be manipulated by the equipment during the assembly process. It therefore needs to be possible to define their interrelations.

The next step of the knowledge transformation during assembly system design is the assembly process specification. This domain conceptualisation and its more specific relationship to the product model have been defined and discussed in the next chapter.

5 Assembly Process Domain Ontology

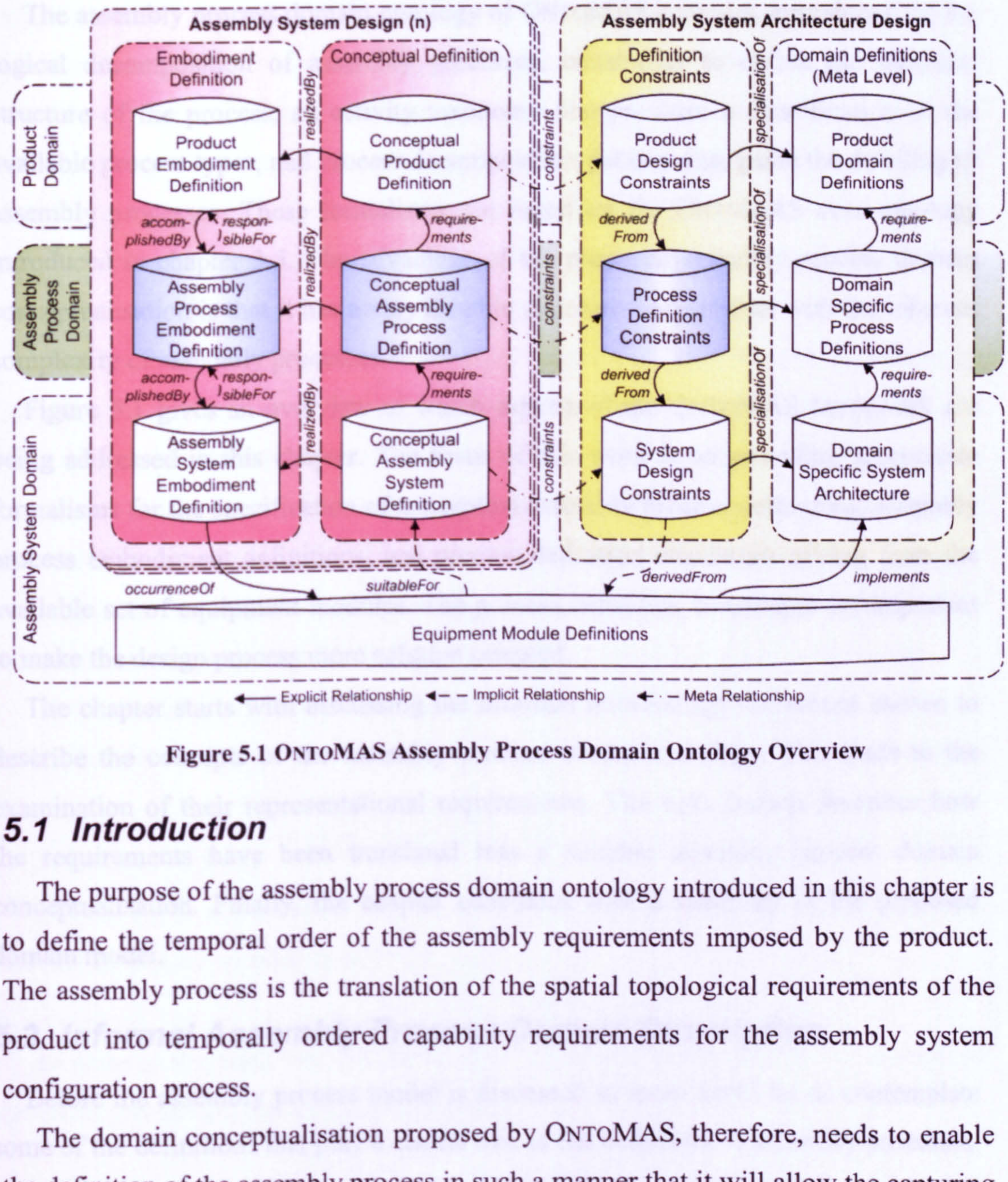


Figure 5.1 ONTOMAS Assembly Process Domain Ontology Overview

5.1 Introduction

The purpose of the assembly process domain ontology introduced in this chapter is to define the temporal order of the assembly requirements imposed by the product. The assembly process is the translation of the spatial topological requirements of the product into temporally ordered capability requirements for the assembly system configuration process.

The domain conceptualisation proposed by ONTOMAS, therefore, needs to enable the definition of the assembly process in such a manner that it will allow the capturing of all the relevant requirements for the definition of suitable assembly systems. The information contained in the product model needs to be presented in such a manner that it contains all the information required to specify the right equipment to facilitate

the assembly process. Most of the current assembly processes definition methodologies focus on a high level of abstraction. They are normally only focusing on the definition of the required task sequences. The information which can be expressed on such high level of abstraction is however far from sufficient to define assembly workstation configurations if the desired equipment granularity is below workstation level.

The assembly process domain ontology of ONTOMAS provides formalisms for the logical decomposition of assembly processes, constraints to define the temporal structure of the process, an activity taxonomy that provides a classification of the available process types, and process decomposition patterns that guide the detailing of assembly processes. These formalisms are based on the ONTOMAS core ontology introduced in chapter 3.5. The advantage of the proposed assembly process domain conceptualisation is that it has a very flexible structure that can deal with the inherent complexity of assembly processes.

Figure 5.1 gives an overview of which aspects of the ONTOMAS framework are being addressed in this chapter. The focus of this work is on providing appropriate formalisms for the specification of conceptual assembly process definitions, assembly process embodiment definitions, and process definition constraints arising from the available set of equipment modules. The process definition constraints are important to make the design process more solution oriented.

The chapter starts with discussing the informal terminology definitions chosen to describe the concepts of the assembly process domain ontology. This leads to the examination of their representational requirements. The next section describes how the requirements have been translated into a suitable assembly process domain conceptualisation. Finally, the chapter concludes with a summary of the proposed domain model.

5.2 Informal Assembly Process Domain Description

Before the assembly process model is discussed in more detail let us contemplate some of the definitions that play a central role in the definition of assembly processes. This will help to develop a better understanding of the considerations that went into the choice of terminology. Terminology of course is always relative since it can be interpreted and used differently by different domain experts. This aspect, however, can be built into any domain ontology by defining a thesaurus structure (see chapter

3.5). The definition of a thesaurus that defines the relations between different domain specific terms or concepts has not been part of the reported work. The chosen modelling formalism does however allow a thesaurus structure to be added at a later point.

It needs to be stressed that the chosen terminology is not instrumental for the proposed modelling and assembly process decomposition approach. What is important however are the underlying principles of the model including defined levels of hierarchy, concept classifications, structural definitions, and a formalism for the definition of the alternative decompositions. It is of course beneficial for the understanding of the proposed methodology to choose the used terminology carefully since it does significantly aid the understanding. The chosen terminology has therefore been based in most parts on widely accepted terms within the area of general assembly.

NIST is defining activities as the central concept of their Process Specification Language (Schlenoff, et al. [113]). Each activity has a defined beginning and end point which could possibly be at infinity. Furthermore, activities can be broken down into temporally ordered sub-activities. This definition of activities is consistent with the overall ontology structure defined above (see chapter 3.5). Activities can therefore be postulated to be the most general concept of a process specification. This postulation is supported by the Oxford English Dictionary definition of activities:

> "Anything active; an active force or operation" OED [90]

The following definition of activities can be made based on the above discussion:

Definition: Activity

"An activity is the formal definition of change over time with a defined beginning and end caused by an actor or actors."

An actor is anything that facilitates the occurrence of the activity, for example a device or mechanism but also a human being.

The fundamental notion of a process is defined by the Oxford English Dictionary as:

> "A continuous and regular action or succession of actions, taking place or carried on in a definite manner, and leading to the accomplishment of some result; a continuous operation or series of operations." OED [90]

In this definition of a process it seems possible to substitute the notion of action and operation, which will be discussed later, by the more general concept of activities that has been introduced above. This will have two implications for a formal model of the assembly process. First, any process could be defined as an activity or set of activities. Second, an activity would be the generalisation of at least actions and operations. Homem de Mello and Sanderson [55] more specifically define the assembly process as a succession of tasks joining subassemblies to form larger subassemblies. It becomes clear from this definition that the goal of an assembly process is the assembling of a set of components and therefore the completion of a product.

The following slightly restricted definition of a process will be used throughout this work which limits processes to be a set of activities but not just one atomic activity.

Definition: Process

"A process is a temporally ordered set of activities with a defined goal."

More specifically a manufacturing process can be defined as:

Definition: Manufacturing Process

"A manufacturing process is a process that has the creation or partial creation of a product as its goal."

An assembly process is then:

Definition: Assembly Process

"An assembly process is a manufacturing process that has the putting together of components as its goal."

It can be argued that the goal of an activity or process is a specific state change as defined in state-transition-diagrams. This leads to the conclusion that any activity needs to have at least one definable start and end state. States represent objects at points in time at which their dynamic attributes are clearly defined. For instance, the relative location and orientation of a part is changing during an assembly process but is definable at the beginning and end of the process. The states only represent information that is already contained in other form in the activity model and they therefore do not need to be an independent part of the actual model since it can be derived. For the specification of the requirements for an activity it is advantageous to express them through the desired state transition which can be defined through an input and output state. For the states during the process there should be a procedure/method associated to the model that can derive the state conditions before and after any given process. States are defined as:

Definition: State

"A State is the specification of the dynamic characteristics an object or group of objects has at a specific point in time."

An assembly sequence is a specific type of process that has only sequentially ordered activities. This type of process is defined as:

Definition: Sequential Process

"A sequential process is a special type of process with an only sequentially ordered set of activities."

Throughout the assembly literature different terms are used to define hierarchical levels for the assembly process structuring. However, a convergence towards the three levels proposed in this work: task, operations, and actions, can be observed where tasks define the highest level and actions the lowest level in the activity hierarchy.

Actions are fundamental activities that provide the building blocks for all processes. The Oxford English Dictionary defines an action as:

> "The way in which an instrument acts; also concr., the arrangement or mechanism by which this is effected." OED [90]

From the above definition of an action it is clear that there is a close link between one actor (instrument) and the action. Combined with the postulation that an action provides the elementary activities of the model an action is defined as:

Definition: Action

"An action is a fundamental activity that can be performed by an actor without the goal to directly influence an object."

The object in the definition is the entity that is involved in the activity other than the actor. The reason for this condition is that an action should be generic and not depending on specific types of objects. Based on this assumption it should be possible to apply an action to any other object without changing the meaning or intention of the action.

Furthermore, an action should be only those activities that do not directly change the state of any of the objects related to the product. This means that an action alone should not be sufficient to change any part attributes or establish liaisons. Taking the example of actions in the assembly domain, a motion action does not actually move any component unless combined with a hold action. The hold action in turn establishes a relationship between the moving device and the component but does not establish a relationship between different product relevant components.

The above action definition also provides a clear link to the functional capability definition of an actor as will be demonstrated later on (see chapter 6). This is very advantageous since the equipment selection process is looking to find suitable pieces of equipment that match specific process requirements. If there is a clear link between the requirements definition in terms of activities and the capability definition of equipment in terms of functions then this will make the matching much more comprehensive and unambiguous.

Operations are generally defined by Oxford English Dictionary as:

> "An act of a practical or technical nature, esp. on forming a step in a process"; "A particular form or kind of activity; an active process: the discharge of a function" OED [90]

This definition conveys a sense that an operation is a part of something bigger and at the same time achieves some goal. This is why operations in this model denote the middle level between actions and tasks. The operations are temporally ordered sets of actions and sub-operations and are therefore processes. This definition is supported by Rampersad [99] who defines assembly operations as a sub-activity of a task. In his definition an operation can also be broken down into sub-activities that define it.

Operations have, contrary to actions, the goal to change concepts that are part of the product definition, as for example components, subassemblies, or liaisons. That means that the result of an operation could for example be the establishment of a liaison between two components or the changing of component attributes, as for example, its position. An operation can therefore be expressed as a state transition of one or more of the concepts that are part of a target product. Operations are defined as follows to express these characteristics:

Definition: Operation

"An operation is a process that facilitates a state change of entities that are part of a product."

The best example of operations in the assembly domain is the assembly operation itself. The primary state transition that is facilitated in an assembly operation is the establishment of at least one liaison. The assembly operation can also involve secondary state transitions of the components being assembled such as change of their position or other properties.

Tasks define the highest level of activities in the proposed assembly process hierarchy. Tasks can be defined on several hierarchical levels themselves. The lowest level tasks are built from a set of temporally ordered operations. Oxford English Dictionary defines a task as:

> "A piece of work imposed, exacted, or undertaken as a duty or the like; originally, a fixed or specified quantity of labour or work imposed on or exacted from a person; later, the work appointed or assigned to one as a definite duty." OED [90]

The concept of a person can be substituted by the concept of an actor which, in the wider sense of automation, could be a machine or autonomous piece of equipment that replaces human operators. What is also clear is that a task encompasses the notion of a clearly specified quantity of work towards a specific goal. In the case of manufacturing the goal is the completion of a product. Homem de Mello and Sanderson [55] specifically define an assembly task as the joining of two sets of parts or alternatively as the establishment of at least one assembly liaison. This is a well defined step during the assembly of a product and therefore supports the above argument for a clearly definable goal.

Following this discussion, tasks are defined as:

Definition: Task

"A task is a process that facilitates a clearly definable portion of work towards the completion of a product."

A portion of work is, for example in assembly, the putting together of two components from the state they are supplied to the assembly process to the state at which they move to the next assembly stage or reach the end of the assembly process. In the simplest assembly case, the insertion of a peg in a hole, the assembly task would encompass supply operations of the two components involved, the actual assembly operation, and the removal operation of the resulting assembly. Optionally the task could also involve up to three handling operations depending on the physical layout of the equipment involved.

5.3 Domain Model Requirements

This section defines the requirements for the proposed process domain ontology. The representation of quite specific activities that need to be performed by the equipment is required for the effective selection, configuration, and control of equipment modules at sub-workstation level. These activities need to be expressed in such a way that they are quite closely related to, if not the same as, the formalism that describes the available equipment capabilities. This is critical for the matching of the required assembly activities against the existing equipment capabilities.

For the equipment selection and evaluation it is important to understand the required technical as well as temporal constraints. The technical constraints for the

assembly specific activities are normally directly derived from the liaisons that need to be established. The time constraints are determined from the overall project requirements including the desired output rate, number of shifts, permissible failure rate, etc. This information needs to be derived from the product model and has to be represented in the assembly process model.

Not all the information from the product model needs to be translated but only the liaisons between the components. The characteristics of the components themselves need to be directly available for the equipment selection. For example, the specification of a gripper or feeder heavily depends on the geometric definition of the components to be held or supplied.

The process domain model requires an additional dimension that goes beyond providing the formalisms for the representation of assembly processes. It also needs to provide the means to define and constrain how the high level assembly tasks are decomposed to derive the required higher level of detail. This formalism should be able to take different levels of abstraction, different levels of hierarchy, and temporal relationships between the activities in a process definition into account.

In the following subsections the more specific requirements for the different aspects of the domain model will be discussed in more detail.

5.3.1 Activities

Activity concept provides traditionally the building blocks of the assembly process model (Schlenoff, et al. [113], Zeigler, et al. [141], WPDL [137]). The *Activities* define what an assembly system needs to do and in what order. Activities can be compared with active functions that need to be performed by an actor to achieve a specific objective. This is important for the equipment selection problem, since the correlation between the required activities and the provided capabilities need to be well understood. The matching can only be achieved if both the required assembly activities and the provided equipment capabilities can be related directly to each other or to a common intermediate concept. The function concept is therefore providing an ideal concept for the matching (see chapter 6 for a more detailed discussion).

For the selection of the right equipment it is important to define the technology parameter of the individual activities. These should become more equipment specific the lower they are in the activity hierarchy. For example, for the insertion of one component into another it is important to understand the required relative motion

between the two components, how much force is needed to perform the motion, and what the necessary relative position and orientation accuracies are between the two components.

Furthermore, the process definition needs to include the specification of the components that participate in the assembly process. This information is required for the selection of equipment that directly handles components. For the definition of this kind of equipment it is often important to know the weight and geometry of the components and also some characteristics such as their role in the assembly, how easy it to orientate and feed them etc. For further discussion of the required component attributes see chapter 4.

Since the description of the required assembly process is already the first abstract definition of the assembly system, it would be very helpful to have a method already at this point to obtain the first predictions of the likely cost and cycle time implications resulting from the choice of process. This would not only help to get a better idea of how realistic the planned cost and cycle time of a system is, but also help to choose the more promising process definitions when there is a choice between two or more possible system solutions.

5.3.2 Logical Structure

A high amount of detail is required for the sufficient representation of the assembly process (see discussion above). This calls for a mechanism to deal with complex representations. A hierarchical definition provides such mechanism as has been outlined in chapter 3. There also needs to be a formalism to define and manage alternative process structures within the process hierarchy. There will always be alternative ways of defining the assembly process and it will not always be clear which one is the better. Therefore, these alternatives need to remain part of the assembly process representation until a more informed decision can be made. Furthermore, there are also cases in which earlier decisions need to be revised and it would be more effective if the alternatives are still known.

It would also be very advantageous for the matching of activities against equipment capabilities if there were a correlation between the distinct levels of the process model and the hierarchical levels of the equipment structure. This would allow the search for the right piece of equipment to be narrowed down dramatically to just the relevant hierarchical level.

The process domain knowledge model does not only need a formalism for the representation of different hierarchical levels – it also needs a formalism that allows the gradual increase of an activity's specification during the definition of an assembly process. It should be possible to only define an activity very abstractly at the beginning of the assembly process definition and make it more specific in the further cause of the process definition. For example, at the beginning of the process specification it might only be possible to deduce that an assembly operation will be needed but not yet its more specific type. Later on more specific information might become available that allows the assembly operation to be more precisely defined as an insertion operation.

5.3.3 Temporal Process Topology

The definition of an assembly process is a temporally ordered set of activities as has been discussed above. It is therefore important to represent the temporal relationships between the different activities in the process definition. The formalism used to define this relationships needs to be able to represent all different ways in which processes could take place. These include sequential, parallel, and recurring activities or sets of activities. Sequential activities are sets of activities which successively start after the previous one has ended (see Figure 5.2 a). Parallel activities are all those activities that could occur at the same time (see Figure 5.2 b). Recurring activities or activity loops are those activities that can or have to happen more than once throughout the assembly process (see Figure 5.2 c).

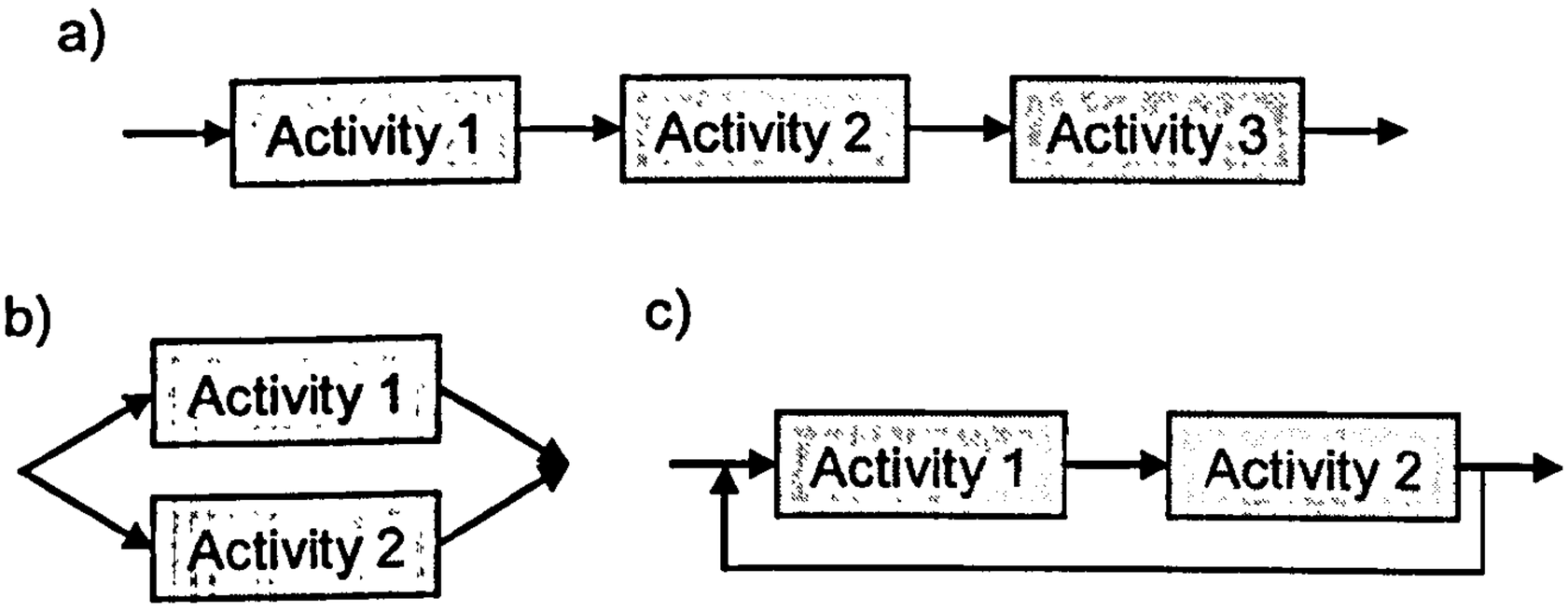


Figure 5.2 Temporal activity structures a) sequential; b) parallel; and c) loop

5.3.4 Process Decomposition

A structured formalism is required for the assembly process decomposition that will guide the effective breakdown of assembly processes into more detailed lower

level activities. Kitamura and Mizoguchi [62] proposed an And-Or-Graph for the decomposition of functions into sub-functions to deduce the higher level capabilities of equipment based on their elementary functions. A similar approach can be used for the decomposition of process requirements.

For the effective decomposition it is important to understand how the different types of activities could be broken down into sub-activities and how they are temporally related to each other. The decomposition definition needs to be based on information that is available at the higher level. This could for example include the activity specification, the product specification, or some already defined related equipment.

During the equipment configuration a very similar if not the same formalism should be applied for the reverse process of synthesising higher level functional capabilities from lower level functions. The relevant equipment functions are closely related to the activity definition in the assembly process domain. Therefore, the same formalism should ideally be applied to achieve both the decomposition as well as the synthesis of activities/functions.

5.4 Formal Assembly Process Domain Ontology

The assembly process domain concepts which have been informally discussed above are formally defined in this section. The underlying modelling formalism is based on ontology engineering principles and extends the basic ontology discussed in chapter 3.5. The main purpose of the proposed assembly process domain ontology is to provide an effective knowledge framework for the specification and decomposition of assembly processes.

The assembly process conceptualisation of ONTOMAS provides formalisms that allow the capturing of all assembly system requirements in a temporally ordered fashion. Figure 5.3 shows an overview of the assembly process conceptualisation resulting from the requirements discussed in section 5.3 and the informal terminology discussed in section 5.2. The *Activity* is the core concept of the assembly process domain ontology. It defines the things that need to be done to achieve the assembly of a product or set of products.

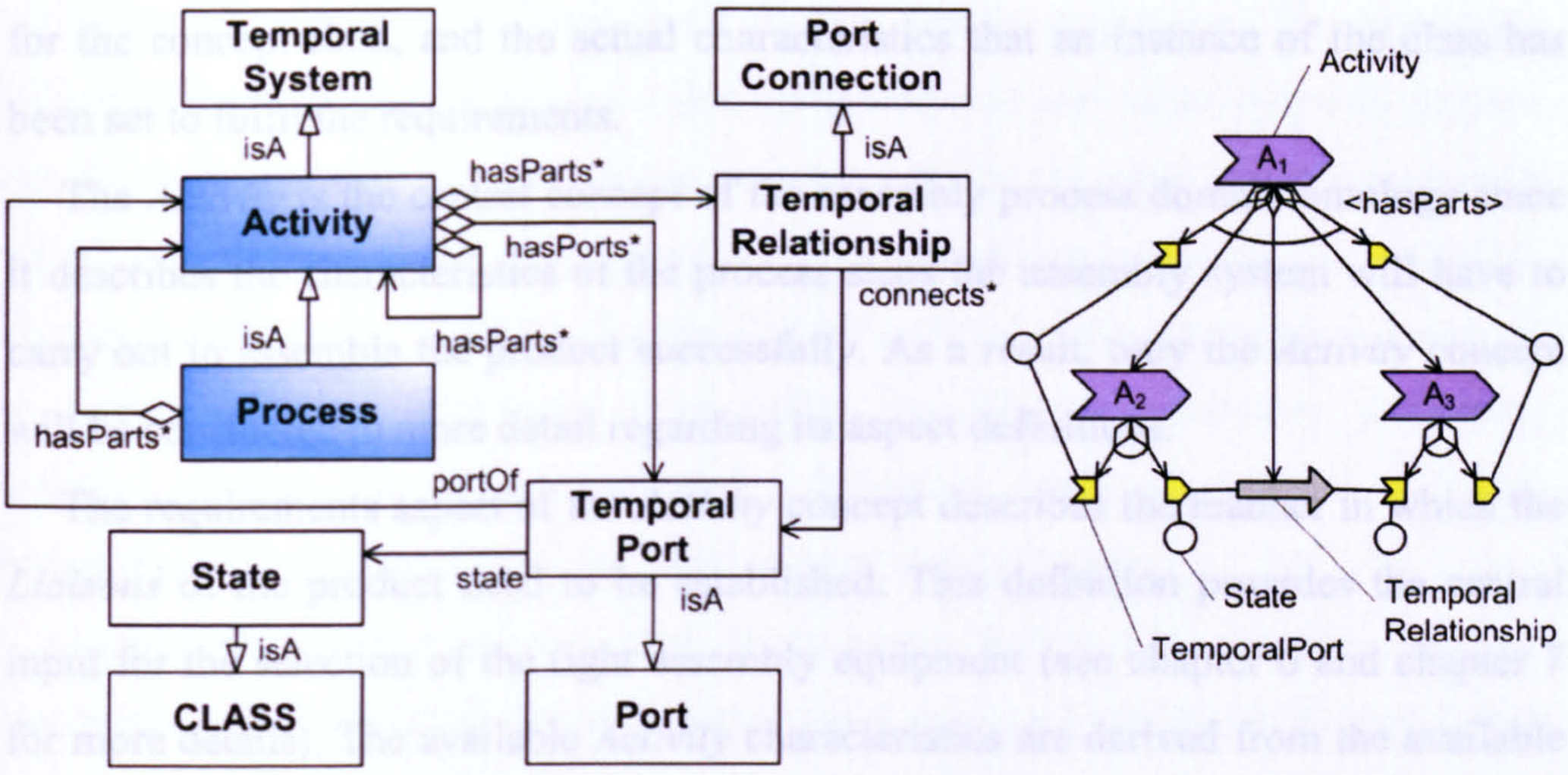


Figure 5.3 Assembly Process Domain Conceptualisation Overview

One of the underlying assumptions for the assembly process domain ontology is that *Activities* can be treated as *TemporalSystems* that has only a finite number of specific *Ports* which define the possible interactions with other *Activities* particularly in regards to their temporal ordering. Hence the *Activity* concept is defined as extension of the *TemporalSystem* concept and is treated as a state-transition. Consequently, each *Activity* has one or more input and output states. *Processes* are defined as specific subsets of *Activities* which have at least one or more sub-activities.

The topology of the process is defined through the *TemporalRelationship* concept which specifies the temporal order between the different *Activities* occurring as part of a process. They are defined as a specialisation of the general *PortConnection* concept and are consequently restricted to specify the connection between *Ports*. In the case of a *TemporalSystem*, these *Ports* are more specifically defined as *TemporalPorts* since they are only defined in the temporal space. The *TemporalRelationships* are defined as part of higher level *Activities* in the same fashion as *Liaisons* are part of assembly definitions. This is the case since *Activities* as well as *Assemblies* are derived from the general *System* concept and *TemporalRelationships* as well as *Liaisons* are their specific type of *PortConnections* respectively.

All concepts of the assembly process domain ontology are either directly or indirectly derived from the central *CLASS* concept. Consequently, they all inherit the three aspect definitions described in chapter 3.5. The aspect addresses the specification of the requirements for a specific concept class, available characteristics

for the concept class, and the actual characteristics that an instance of the class has been set to fulfil the requirements.

The *Activity* is the central concept of the assembly process domain ontology since it describes the characteristics of the process steps the assembly system will have to carry out to assemble the product successfully. As a result, only the *Activity* concept will be considered in more detail regarding its aspect definitions.

The requirements aspect of the *Activity* concept describes the manner in which the *Liaisons* of the product need to be established. This definition provides the central input for the selection of the right assembly equipment (see chapter 6 and chapter 7 for more details). The available *Activity* characteristics are derived from the available assembly equipment characteristics which result from the chosen equipment configuration (see chapter 6). The actual *Activity* characteristics are similarly derived from the actual assembly equipment characteristics once a system has been defined.

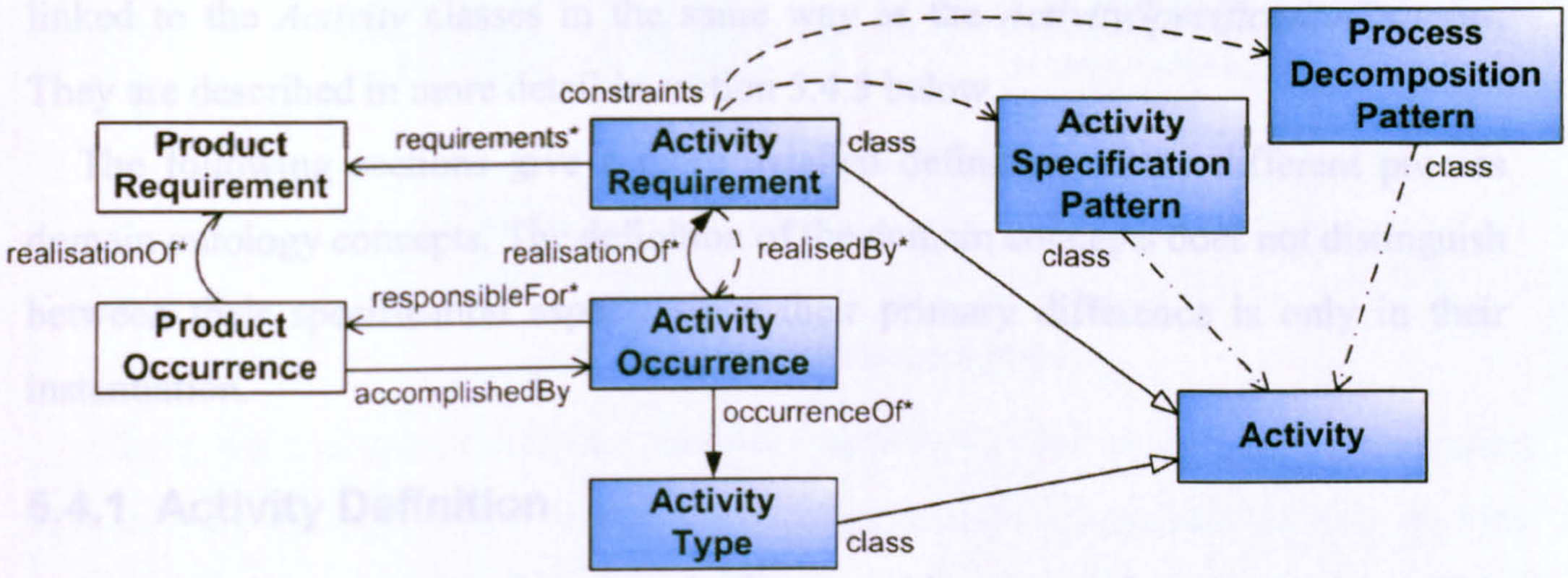


Figure 5.4 Relationships between main Aspects of the Assembly Process Domain Concepts

Figure 5.4 shows the aspect relationships between the central concepts of the product and assembly process models. The *ActivityRequirement* concept defines the required characteristics for the *Activities* through the <requirements> attribute. The concept allows the selection of an appropriate *Activity* class for its specific requirements with the <class> attribute. The *ActivityOccurrence* concept specifies the actual instance of the *Activity* that realises the requirements specified by the *ActivityRequirements* concept. The <responsibleFor> attribute associates the *ActivityOccurrence* with the actual part of the product model it is responsible for establishing. The main objective for the definition of assembly specific *ActivityRequirements* and *ActivityOccurrences* is the establishment of *Liaisons* that define the spatial and logical relationship between *Components* in the product model.

ONTOMAS provides the formalisms for an *Activity* taxonomy. The taxonomy defines different classes of *Activities* that could occur as part of an assembly process specification. The tree structure of the taxonomy fulfils the need of the engineering design process to support the abstraction and specialisation during the assembly process specification. The basis for the classification of in the taxonomy is captured by *ActivitySpecificationPatterns*. They define the difference between alternative *Activity* classes. A more detailed definition and discussion of the *ActivitySpecificationPattern* can be found in section 5.4.3 below.

Another engineering process that takes place during the assembly process specification is the decomposition of higher level processes into more detailed lower level descriptions. This process can be assisted and constraint with *ProcessDecompositionPatterns* that capture recurring aggregation patterns and link them to conditions under which they occur. The *ProcessDecompositionPatterns* are linked to the *Activity* classes in the same way as the *ActivitySpecificationPatterns*. They are described in more detail in section 5.4.5 below.

The following sections give a more detailed definition of the different process domain ontology concepts. The definition of the domain concepts does not distinguish between their specification aspects since their primary difference is only in their instantiation.

5.4.1 Activity Definition

The *Activity* is the central concept in the assembly process domain ontology. This has been defined based on the Process Specification Language (PSL) definition from NIST (Bock and Gruninger [10]). Furthermore, *Activities* are defined as specialisations of *TemporalSystems* (see Figure 5.5). That means that the interaction between *Activities* is restricted to the *Ports* an *Activity* exposes to the outside world. Each *Activity* has to have at least two *TemporalPorts* to define their start and end point.

The two fundamental characteristics of the different classes of *Activities* are their <hasParts> and <responsibleFor> attributes. The constraints for the assembly process decomposition are mainly concerned with these two attributes. The <hasParts> attribute replaces the <subactivity> attribute defined in PSL and allows the constraining of the sub-activity types and temporal constraints an *Activity* can or has to contain. The <hasParts> attribute is working in very similar manner to the

<subactivity> attribute defined in the PSL core. The reason for using the <hasParts> attribute is to create a consistent terminology across all the assembly relevant domains, not only the process domain (see chapter 3.5).

The <responsibleFor> attribute defines what types of *CLASSES* are being directly manipulated or influenced by the *Activity*. In the assembly domain this would generally be either *Components* or *Liaisons*. The mechanism for the constraint definition is explained in more detail in section 5.4.3.

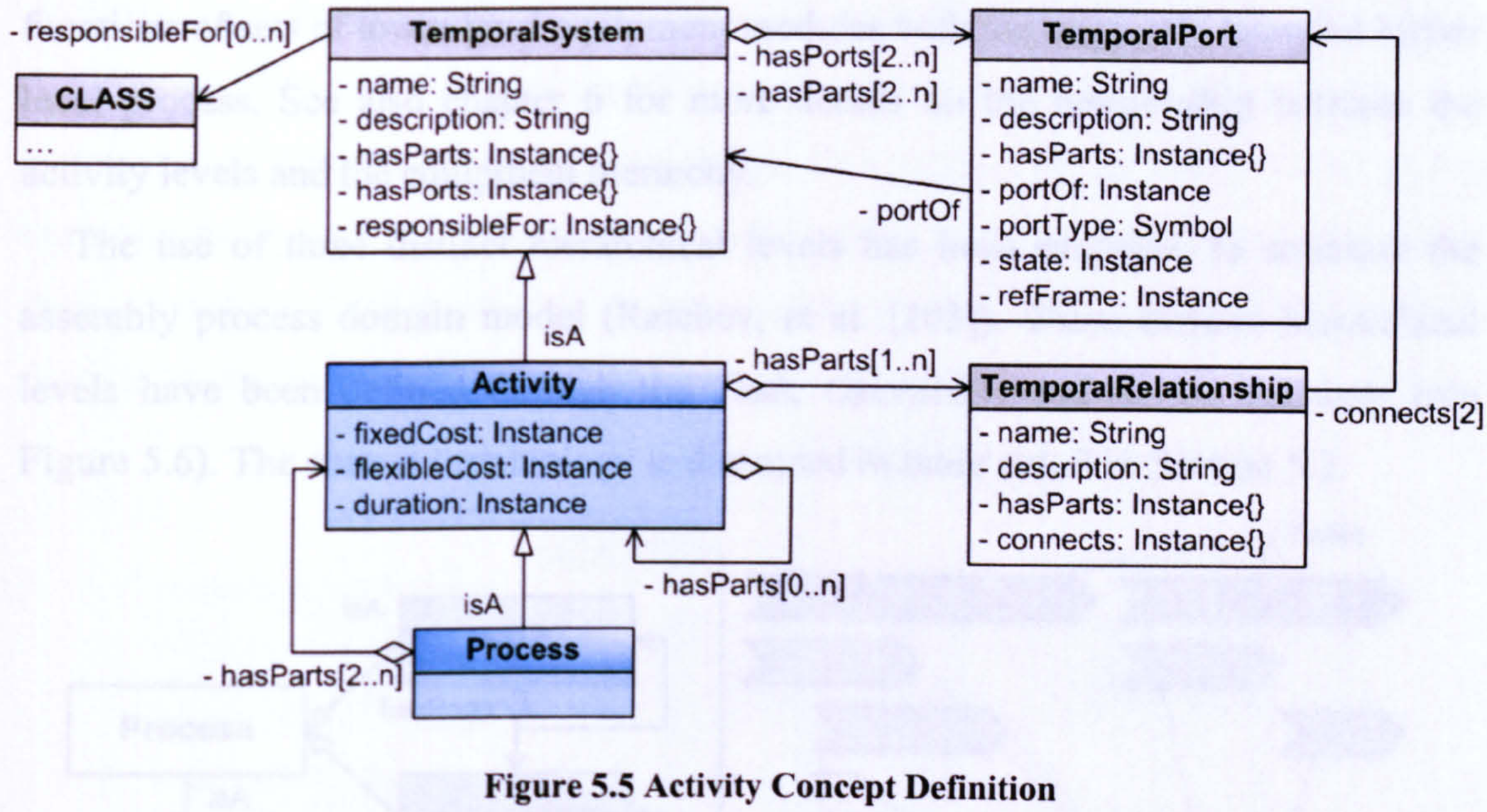


Figure 5.5 Activity Concept Definition

The predicted cost of realising the given activity as well as its predicted duration are defined through values that reflect the nominative magnitude as well as their variation in percent. The cost characteristics are split into fixed and flexible costs to allow for simple cost comparison of different alternative assembly processes.

The *Process* concept is a simple extension of the *Activity* concept with the only added restriction that a process needs to contain at least two sub-activities. The more specific *Activities* like *Tasks*, *Operations*, and *Actions* are defined as extensions of either the *Activity* or *Process* concept. Each of these three types of *Activities* has again more specific sub-types. This will be discussed in more detail in section 5.4.3 and section 5.4.5.

5.4.2 Activity Hierarchy

The *Activities* that constitute an assembly process need to be structured hierarchically to deal with the great complexity of the required highly detailed assembly process model (see section 5.3). Furthermore, there need to be distinct

levels in the activity hierarchy to allow a clear mapping between *Activities* and *Equipment* modules on corresponding hierarchical levels. This will allow a more effective selection of *Equipment* modules with higher levels of capability complexity. For example, if an insertion activity is required it does not necessarily need to be broken down into sub-activities if equipment modules exist whose capability includes the execution of the desired activity at this level of definition. The definition of clearly distinct hierarchical levels also makes it easier to compare synthesised functions of sets of lower level equipment modules with the originally intended higher level process. See also chapter 6 for more details on the relationship between the activity levels and the equipment hierarchy.

The use of three distinct hierarchical levels has been proposed to structure the assembly process domain model (Ratchev, et al. [103]). These distinct hierarchical levels have been defined through the *Task*, *Operation*, and *Action* concepts (see Figure 5.6). The chosen terminology is discussed in more detail in Section 5.2.

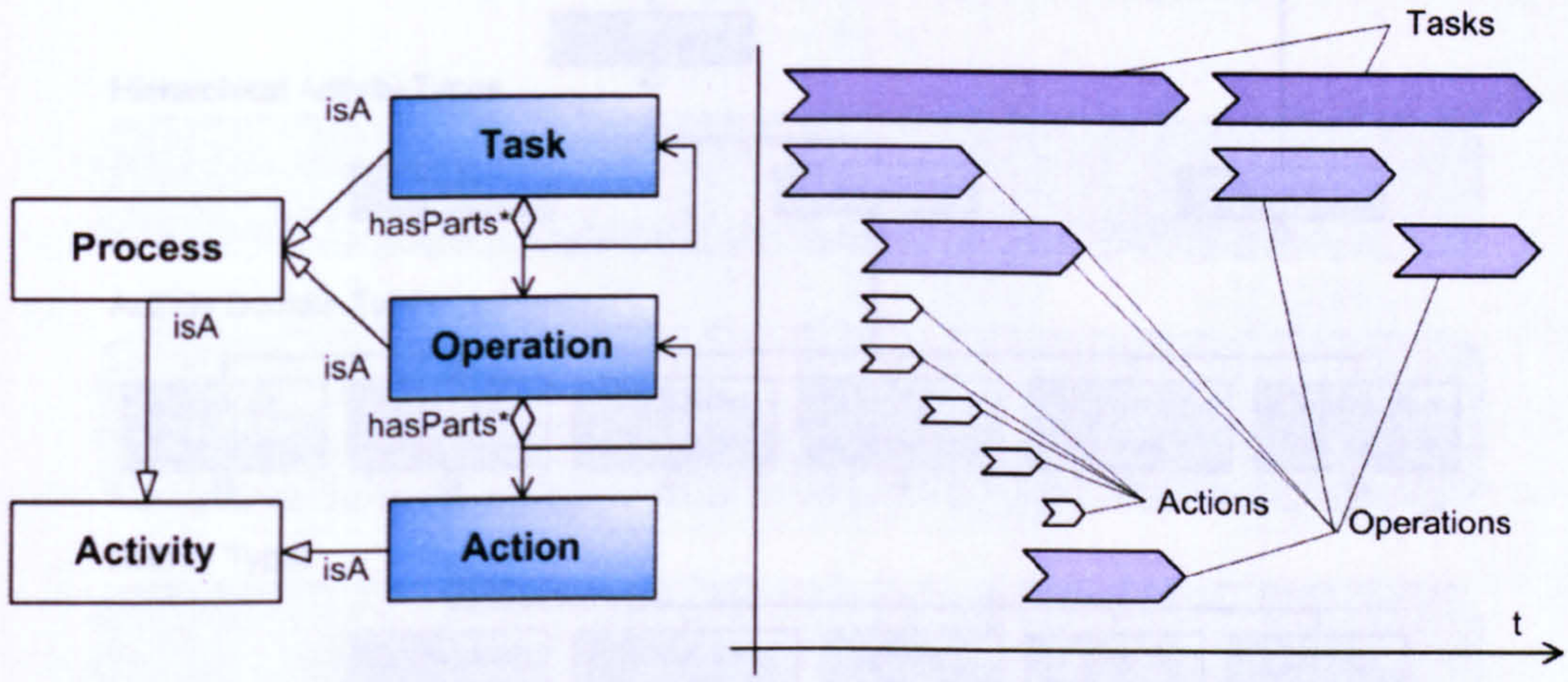


Figure 5.6 Activity Hierarchy Definition

Both the *Task* and *Operation* concepts are sub concepts of the *Processes* concept with specific constraints for their internal structure. Where *Processes* can contain any combination of sub-activities, *Tasks* and *Operations* can only contain very specific types of sub-activities. *Actions* are not *Processes* and define the elementary *Activities* that cannot have any sub-activities.

The activity/sub-activity relationship defined by the <hasParts> attribute of the *Activity* concept has been restricted for *Tasks*, *Operations*, and *Actions* to introduce the required hierarchy between them (see Figure 5.6). This means that *Tasks* can only contain other *Tasks* or *Operations* as sub-activities, *Operations* can only contain other

Operations and *Actions* as sub-activities, and *Actions* cannot contain any sub-activities.

5.4.3 Activity Taxonomy

The activity taxonomy defines how the different types of *Activities* are logically related to each other in terms of abstraction and specialisation. The taxonomy is used to structure the decomposition and specification constraints for the different types of *Activities* in a flexible and comprehensive manner. The use of a taxonomy allows these constraints to be defined on different levels of abstraction which enables the inheritance of the constraints at lower levels. This reduces the definition effort and at the same time promotes a more consistent constraint specification that is less error-prone since high level constraints that can be tested more rigorously are being used as foundation for the specification of lower level constraints.

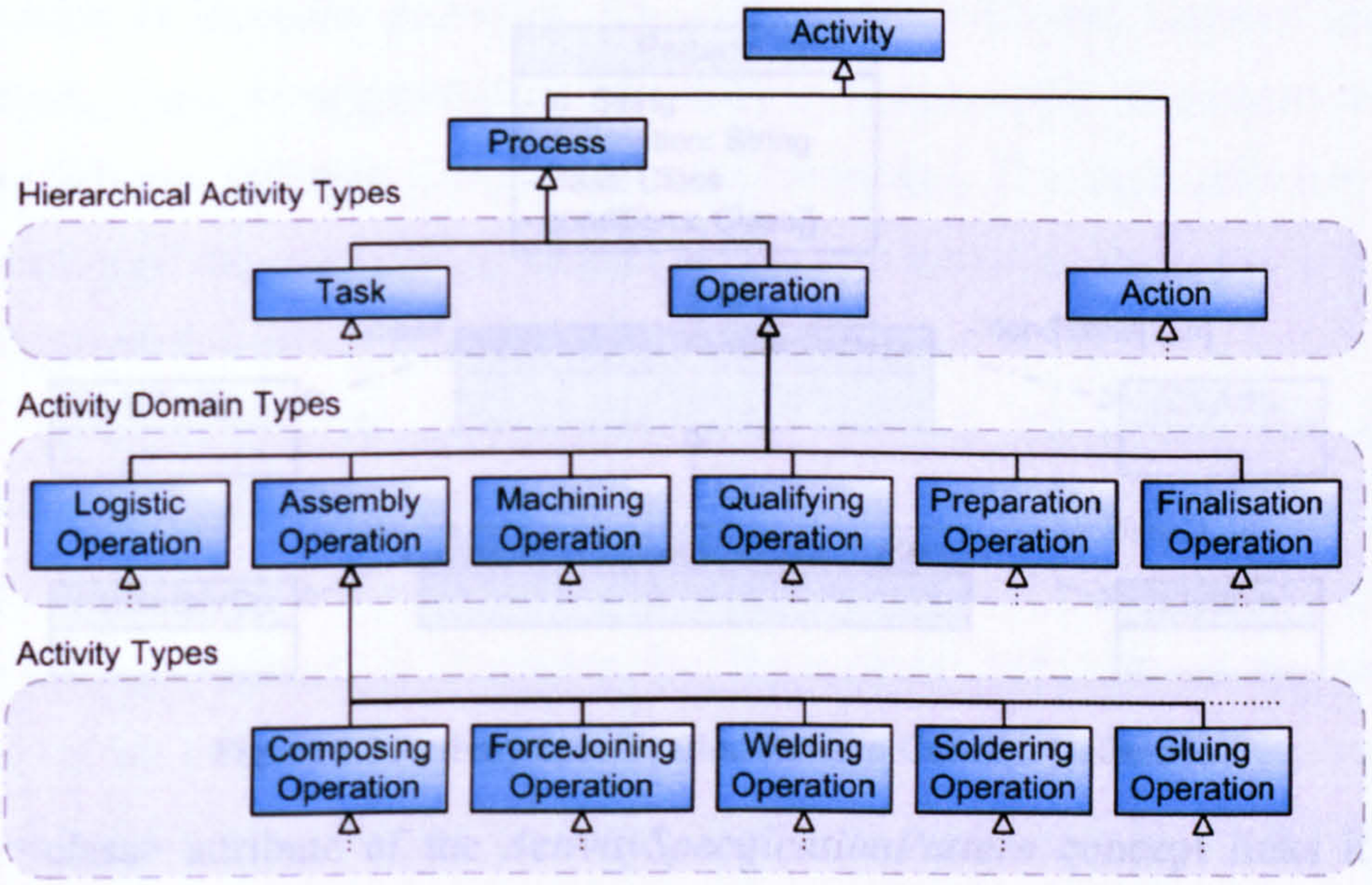


Figure 5.7 Principle Structure of the Activity Taxonomy

The highest level of the taxonomy is defined through the classification of *Activities* into *Tasks*, *Operations*, and *Actions*. The classification provides the concepts for the definition of distinct hierarchical levels in the process definition as discussed in section 5.4.2. Each of the concepts denoting a hierarchical level is further classified to provide distinct activity domain types. There is a close relationship between the defined activity domain types and the equipment domain types defined in chapter 6. The activity domain types are further specialised into domain specific activity types to

allow a meaningful distinction of the different *Activities* needed for the specification of assembly system requirements.

It is important for the classification to be based on qualifiable criteria. The most suitable criteria for the characterisation of different activity types are the different state transitions they define. These are defined as the objectives of an *Activity* and are expressed through its <resposibleFor> attribute. An And-Or-Graph based specification formalism has been defined to prescribe the required and optional targets for an *Activity*.

The *ActivitySpecificationPattern* concept has been introduced to define the constraints placed on the <responsibleFor> attribute of the different *Activity* classes (see Figure 5.8). The *ActivitySpecificationPattern* concept fulfils two roles. It formalises the criteria that distinguish the different *Activity* classes and it provides the means to formally verify if an *Activity* is of a specific class.

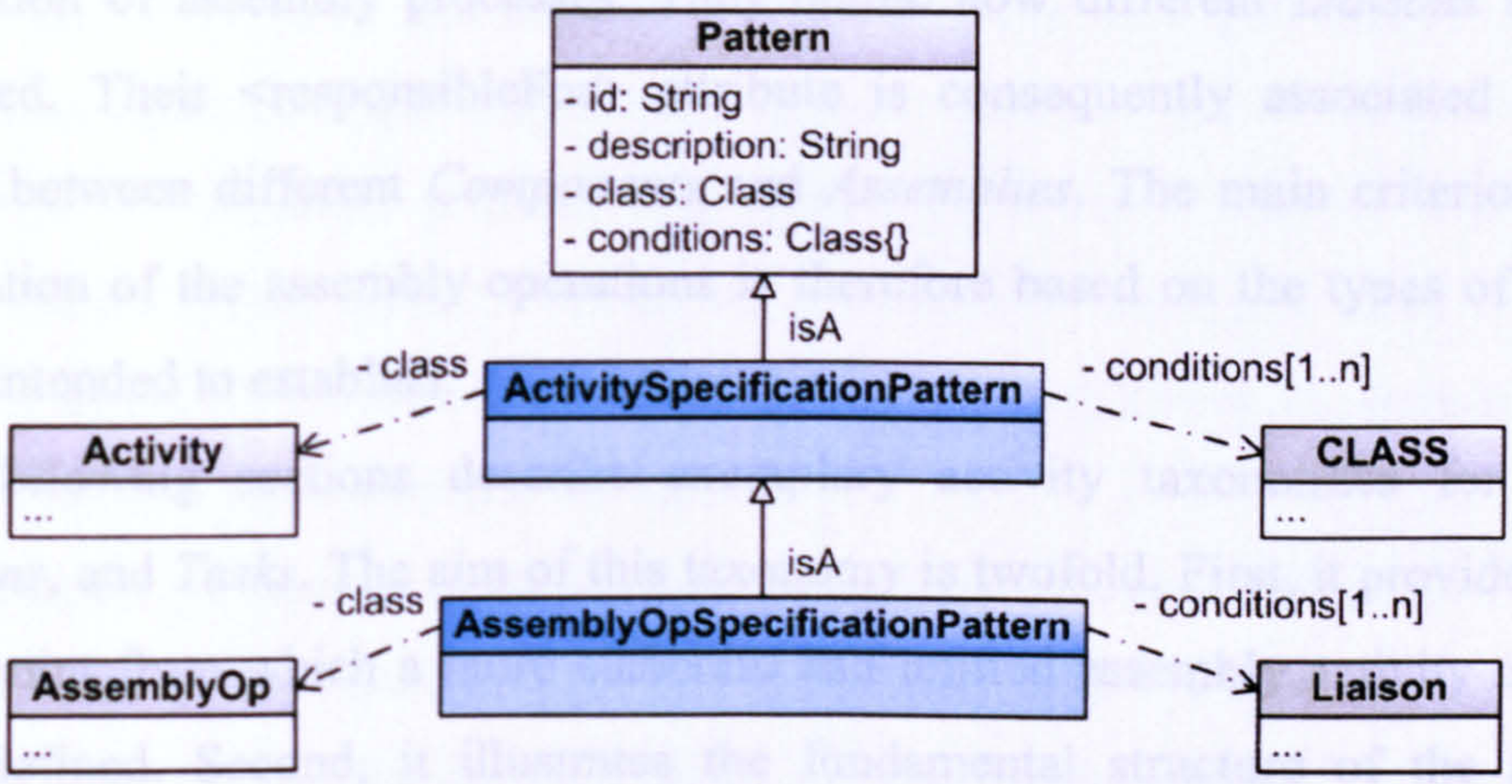


Figure 5.8 Activity Specification Pattern Concept Definition

The <class> attribute of the *ActivitySpecificationPattern* concept links it to the specific *Activity* class for which it is providing the condition specification. The <conditions> attribute defines which types of *CLASSES* are required in the <responsibleFor> attribute of an *Activity* of this class. This formalism has the advantage that it works like an And-Or-Graph. The <class> attribute acts like an OR since more than one *ActivitySpecificationPattern* could refer to the same *Activity*. The <conditions> attribute is defining the AND part by specifying all the required *CLASS* types. The optional and required constraints for an activity type can be determined through the set of *ActivitySpecificationPatterns* that are associated to the *Activity* class. The constraints that are defined in all *ActivitySpecificationPatterns* are always

required and the rest are optional. Figure 5.9 shows an illustrative example of two different *AssemblyOpSpecificationPattern* for the same *AssemblyOperation*.

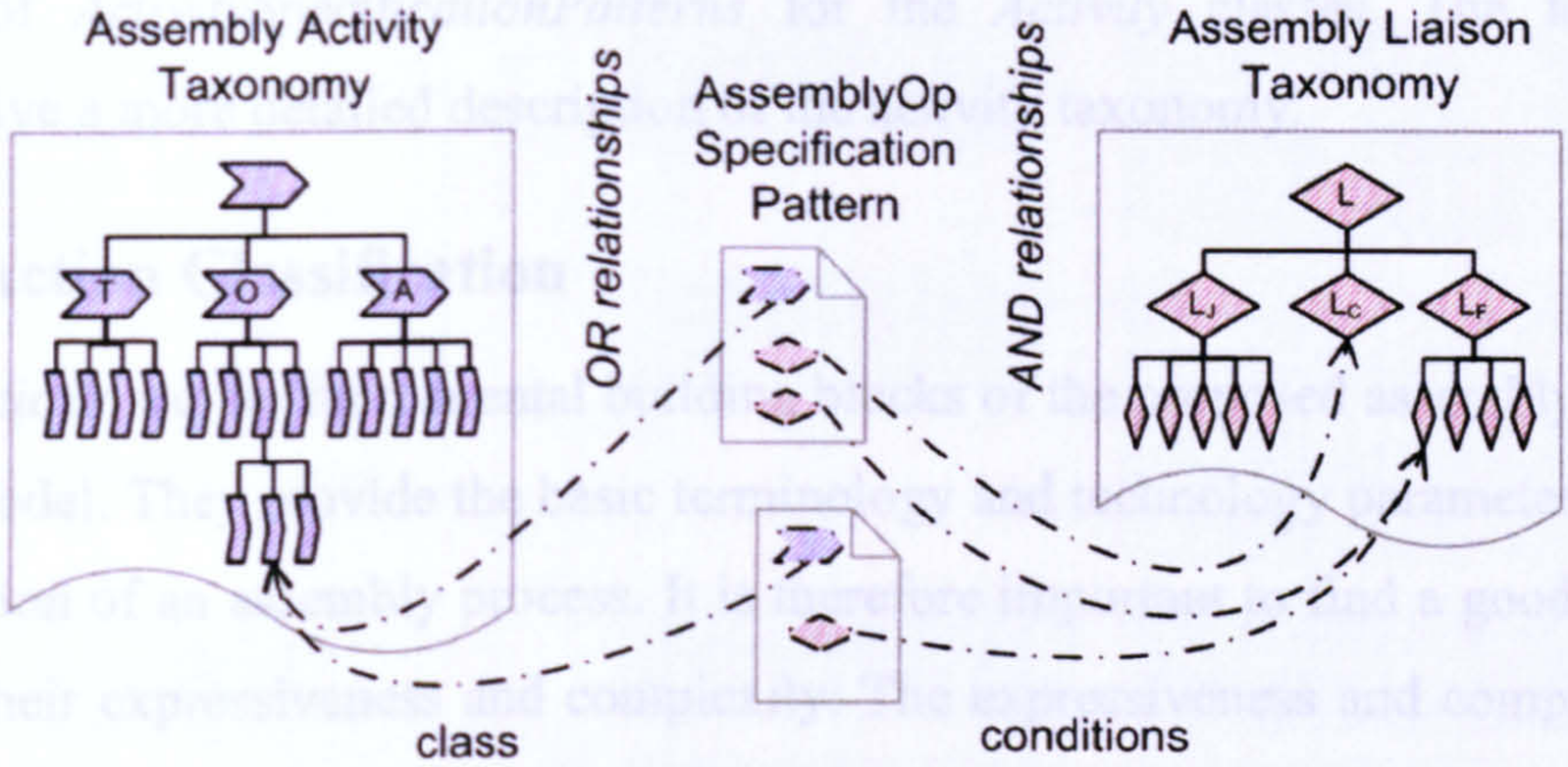


Figure 5.9 Assembly Operation Specification Pattern Illustration

The assembly operation types are the most critical *Activity* classes for the specification of assembly processes. They define how different *Liaisons* are being established. Their <responsibleFor> attribute is consequently associated with the *Liaisons* between different *Components* and *Assemblies*. The main criterion for the classification of the assembly operations is therefore based on the types of *Liaisons* they are intended to establish.

The following sections describe exemplary activity taxonomies for *Actions*, *Operations*, and *Tasks*. The aim of this taxonomy is twofold. First, it provides a good starting point from which a more elaborate and unified assembly activity taxonomy can be defined. Second, it illustrates the fundamental structure of the assembly process that has been used throughout this work. The illustration and discussion of the proposed activity taxonomy for the assembly domain starts with the classification of the *Action* concept since these are the elementary *Activities* upon which the interpretation of assembly processes is established. The classification of the *Operation* concept follows the classification of the *Action* concept. This is the most expressive classification since it needs to provide different types for all the possible different activity target definitions. The taxonomy illustration is concluded with the classification of the *Task* concept which can remain rather abstract since its specific interpretation is based on the *Operations* and *Actions* it contains.

The proposed activity taxonomy does not claim to be complete and neither does it have to be at this point. The definition of a more complete taxonomy would require the involvement of a big proportion of the assembly community. This could be done

at a later point and the resulting new or changed activity types can be easily integrated into the existing taxonomy structure. This only requires the definition, change, or removal of *ActivitySpecificationPatterns* for the *Activity* classes. The following sections give a more detailed description of the activity taxonomy.

5.4.3.1 Action Classification

The *Actions* are the fundamental building blocks of the proposed assembly process domain model. They provide the basic terminology and technology parameters for the interpretation of an assembly process. It is therefore important to find a good balance between their expressiveness and complexity. The expressiveness and complexity of the fundamental model are naturally contradictory; the higher the expressiveness the higher the complexity and vice versa.

A number of exemplary *Actions* has been defined based on VDI 2860 [132], Lotter [77], Pahl and Beitz [94], and Rampersad [99] (see Figure 5.10). The given set of *Actions* is not exhaustive and only covers the most common *Actions* required for the modelling of the assembly process. The *Actions* have been classified into a small number of main types, each of which has a number of subtypes. Each individual *Action* type has its own technological parameters. A *MoveAction* for example has at least one reference point to define the motion.

The technical parameters of the actions are defined in such a manner as to allow a seamless transition from qualitative definitions to quantitative where appropriate. The position of a component for example might need to be changed with a *MoveAction*, but at first the absolute distance is not known. The tendency relative to something else might be known however and could be expressed as above or next to it. This can then be defined as a qualitative specification of the motion which can be later defined more accurately once this information becomes available.

Storing of components and assemblies is one of the integral parts of the logistics of an assembly system. This is due to the requirement for bulk transport and the lack of seamless integration between different parts of the manufacturing process. The *StoreAction* defines the different modes for storing components and assemblies. The *StoreAction* is classified according to VDI 2860 [132] based on the ordering between the components stored. Ordered storage requires the position and orientation of all components to be defined in all 6 degrees of freedom. Unordered storage does not

require the position and orientation of the components to be defined. Partially ordered storage is in-between and requires at least one degree of freedom to be defined.

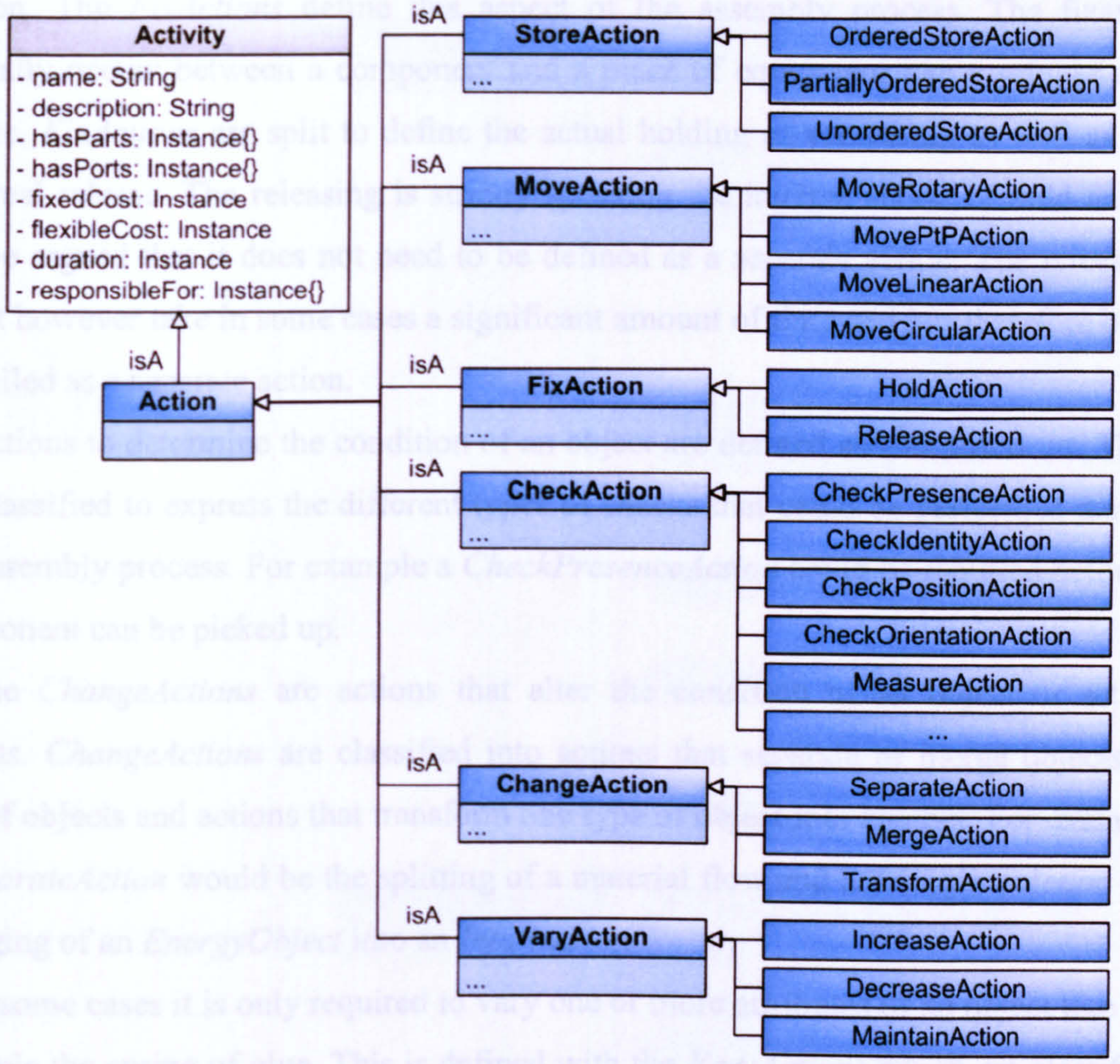


Figure 5.10 Action Classification

The *MoveAction* is one of the most central actions for the definition of an assembly operation. Most assembly operations involve some kind of relative motion between either the two components or one of the components and a tool. The *MoveAction* has been classified into: rotary motion, point-to-point motion, linear motion, and circular motion. This classification considers two perspectives of the *MoveAction*. One is the view from the equipment side which defines motions more fundamentally in terms of translations (linear motion) and rotations (rotary motion). The other is from the product side looking at what motions the components require to facilitate their assembly. These are the point-to-point, linear, and circular motion. The definition of the assembly process is in the first instance based on the product description. It is therefore more meaningful to use component motions to define the required assembly process.

Even so the motion actions are defined as motions of the components; the definition does not define how the components need to be held in order to achieve the motion. The *FixActions* define this aspect of the assembly process. The fixation normally occurs between a component and a piece of equipment like a gripper or a fixture. *FixActions* are split to define the actual holding of something as well as the eventual release. The releasing is strictly speaking the inverse action of hold and it can be argued that it does not need to be defined as a separate action. The releasing might however take in some cases a significant amount of time and has therefore been modelled as a separate action.

Actions to determine the condition of an object are defined as *CheckActions*. They are classified to express the different types of checks that could be performed during the assembly process. For example a *CheckPresenceAction* could be required before a component can be picked up.

The *ChangeActions* are actions that alter the condition of an object or set of objects. *ChangeActions* are classified into actions that separate or merge objects or sets of objects and actions that transform one type of object into another. For example a *SeparateAction* would be the splitting of a material flow and a *TransformAction* the changing of an *EnergyObject* into an *InformationObject*.

In some cases it is only required to vary one or more attributes of an object like for example the curing of glue. This is defined with the *VaryActions* which are classified into increasing, decreasing, and maintaining actions. The *MaintainAction* is also a variation if the natural tendency of the object would be to change its attribute otherwise.

5.4.3.2 Operation Classification

The *Operations* define the intermediate level between *Tasks* and *Actions*. At this level the specialisation of all the different activities could take place during an assembly process. *Operations* tend as a result to be highly specialised with many different levels of classification. The underlying reason is that different types of operations have different possible action compositions. Certain compositions are often recurring during an assembly process and it is therefore reasonable to define them as operation types in their own right.

Operations are to some extent a means to specify and achieve the right action decomposition of the assembly process and to reduce the complexity of the process

definition. Of course they are also used to make pre-selections of equipment types to narrow the search as early as possible.

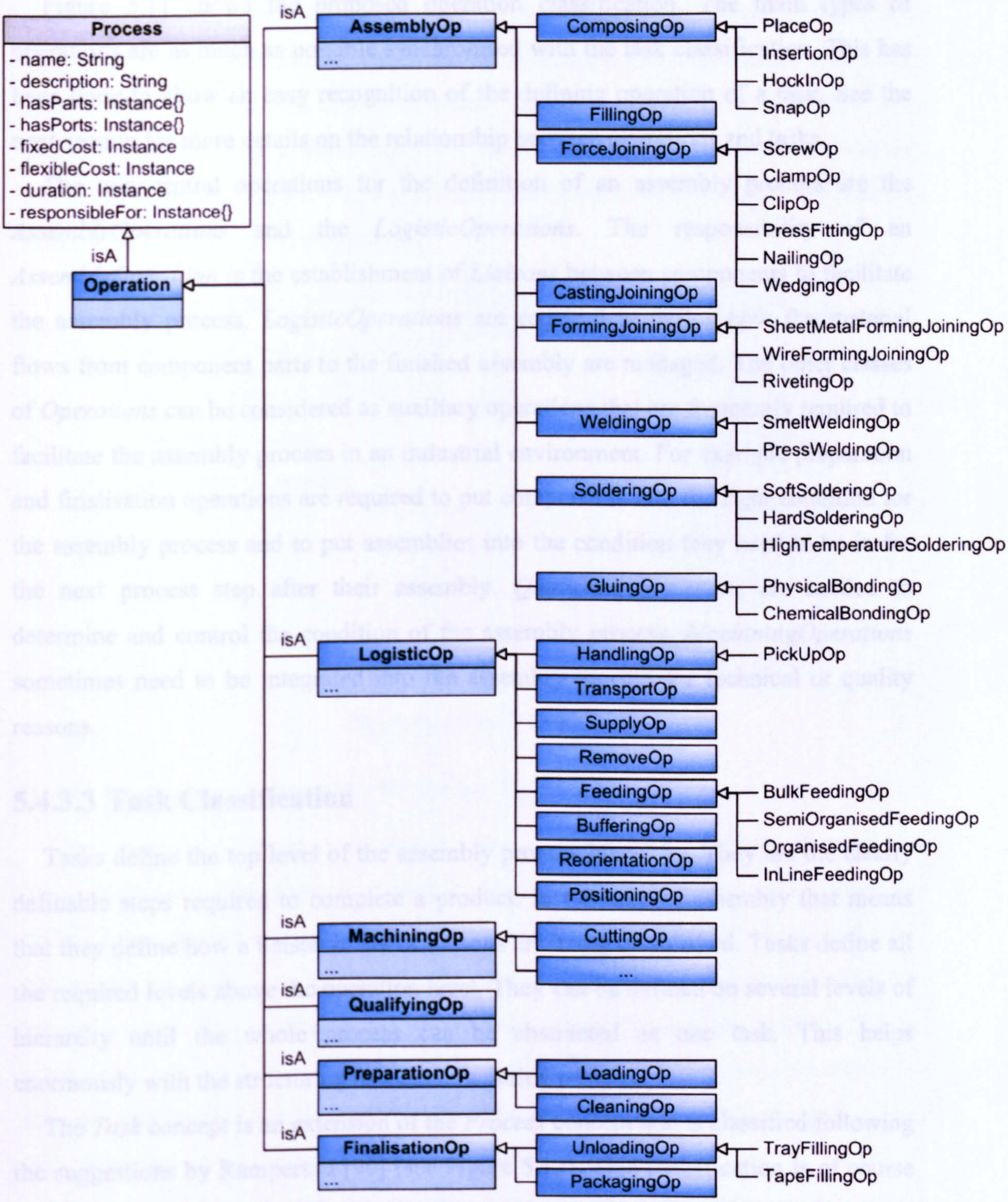


Figure 5.11 Operation Classification

A preliminary classification of operations has been used in this work. The classification is based on the work of Rampersad [99]. The assembly operation classification is based on DIN 8593-0 [28]. The classification does not claim to be

complete but rather provides a good first iteration that is used to demonstrate the proposed methodology.

Figure 5.11 shows the proposed operation classification. The main types of operations are as much as possible synchronised with the task classification. This has been done to allow an easy recognition of the defining operation of a task. See the next section for more details on the relationship between operations and tasks.

The two central operations for the definition of an assembly process are the *AssemblyOperations* and the *LogisticOperations*. The responsibility of an *AssemblyOperation* is the establishment of *Liaisons* between components to facilitate the assembly process. *LogisticOperations* are required to define how the material flows from component parts to the finished assembly are managed. The other classes of *Operations* can be considered as auxiliary operations that are frequently required to facilitate the assembly process in an industrial environment. For example preparation and finalisation operations are required to put components into the right condition for the assembly process and to put assemblies into the condition they need to be in for the next process step after their assembly. *QualifyingOperations* are needed to determine and control the condition of the assembly process. *MachiningOperations* sometimes need to be integrated into the assembly process for technical or quality reasons.

5.4.3.3 Task Classification

Tasks define the top level of the assembly process hierarchy. They are the clearly definable steps required to complete a product. In the case of assembly that means that they define how a liaison or set of liaisons are being established. Tasks define all the required levels above the operation level. They can be defined on several levels of hierarchy until the whole process can be abstracted as one task. This helps enormously with the structuring of a manufacturing process.

The *Task* concept is an extension of the *Process* concept and is classified following the suggestions by Rampersad [99] (see Figure 5.12). This classification is of course not the only possible one and others could be quite as reasonable. This does however not contradict the necessity for a widely accepted classification. Also this does not put the suggested decomposition approach into question since the approach is mostly independent from the actual classification and only requires one to exist.

There is no need for the tasks to be classified any further than the first level. The more detailed distinction between the tasks will be defined through the operations of which they are comprised. *Operations* are on the highest level classified in the same way as the tasks. Each *Task* of a specific type has at least one *Operation* of the same type as a sub-activity. This operation defines the more specific attributes of the tasks and allows its distinctive characteristics to be defined. *Tasks* are not very constrained due to their general nature. Their more specific breakdown is based on the relevant product and equipment characteristics.

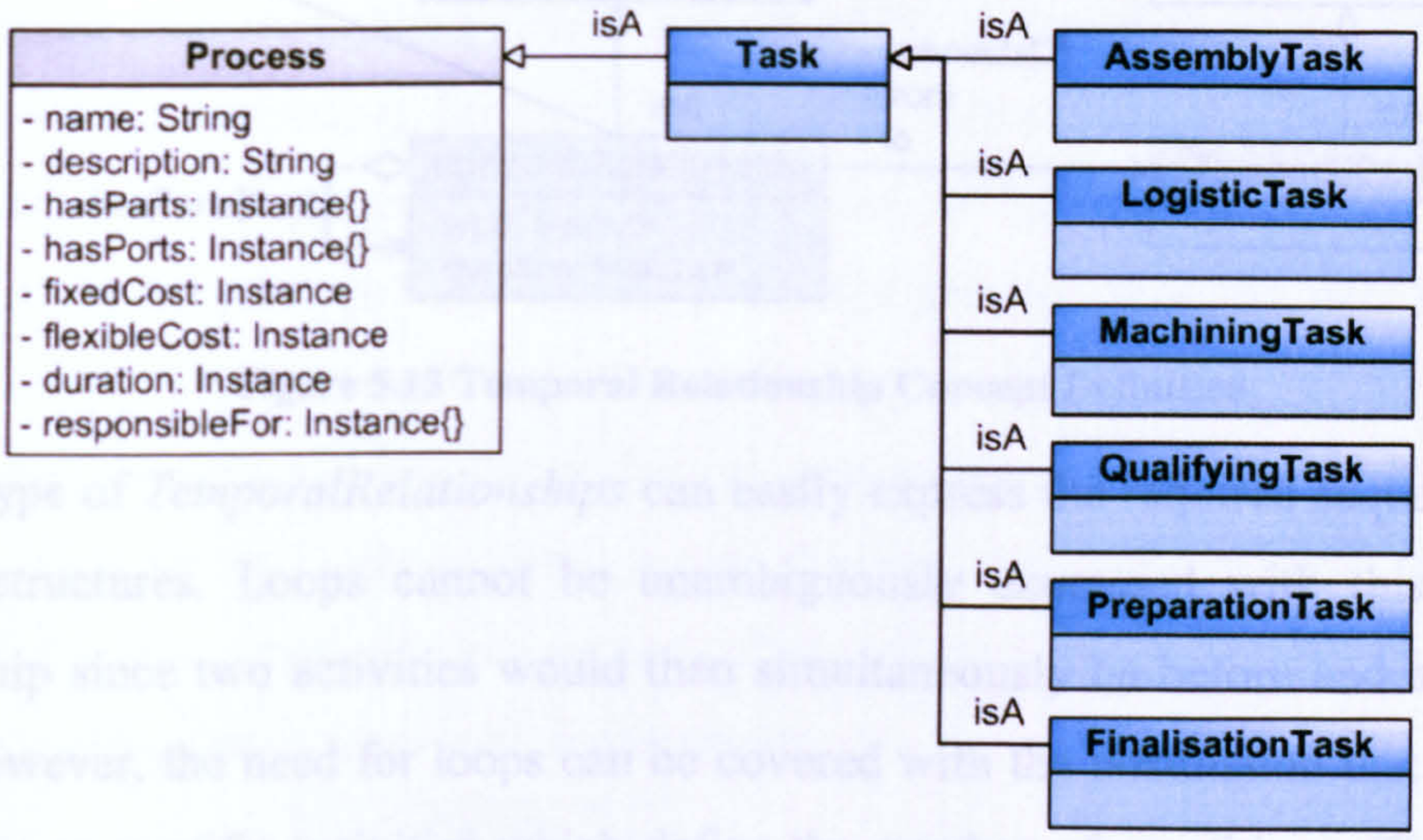


Figure 5.12 Task Classification

5.4.4 Temporal Assembly Process Topology

The temporal constraints between the activities are defined through the *TemporalRelationship* concept shown in Figure 5.13. The *TemporalRelationships* define how activities are temporally ordered between each other. They are defined as extensions of both the *PortConnection* and the abstract *OneDirectionalRelation* concepts since *TemporalRelationships* are both directional and restricted to connect ports. The <from> and <to> attributes inherited from the *OneDirectionalRelation* concept specify the direction of the *TemporalRelationship* by linking to the *Ports* defined as part of the <connects> attribute.

The *TemporalRelationship* can both be defined as symbol and as temporal value to capture different levels of specification detail. The symbolic definition allows the qualitative specification of the temporal order. The symbols used for the qualitative specification are: 'before', 'after', and 'equal'. The value-based specification defines the quantitative ordering of the activities. During the definition the qualitative

specification is much more flexible since activities can be added and removed without having to redefine all the relationships between other processes. With increasing level of detail it becomes more reasonable to determine some first estimation of the times between specific processes. The value is captured as normative magnitude and the variation in percent.

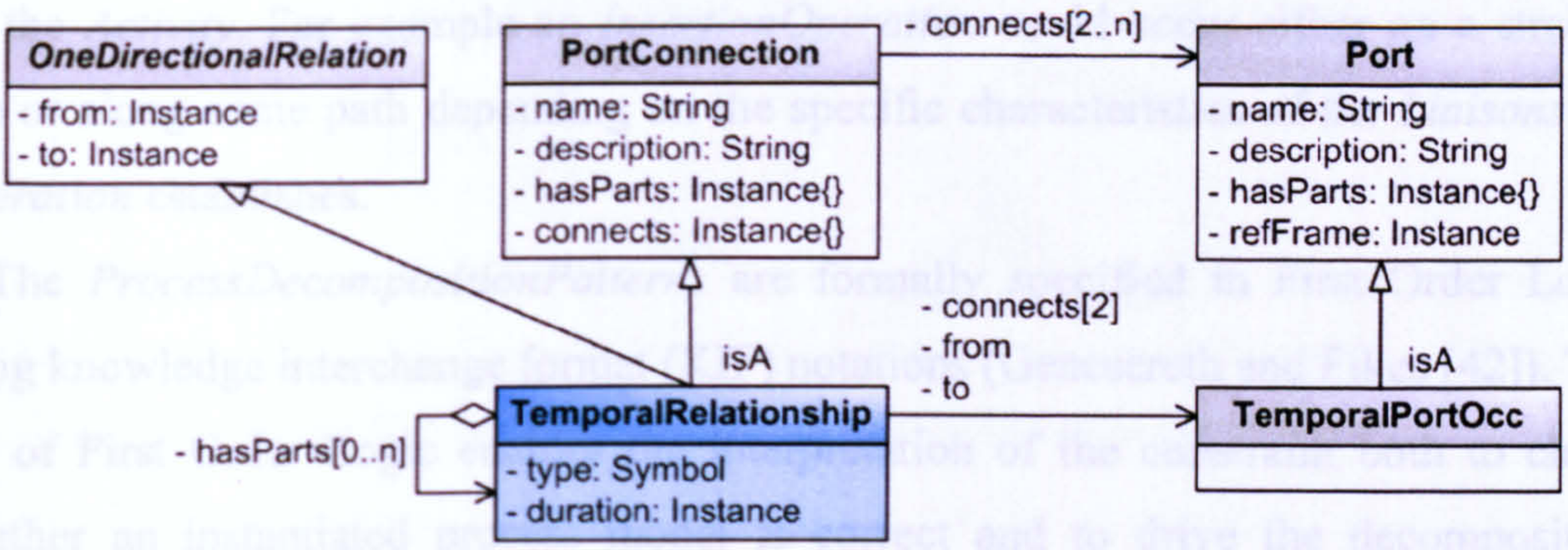


Figure 5.13 Temporal Relationship Concept Definition

This type of *TemporalRelationships* can easily express the required sequential and parallel structures. Loops cannot be unambiguously expressed with this type of relationship since two activities would then simultaneously be before and after each other. However, the need for loops can be covered with the postulation that they can only occur as specific activities which define the number of repetitions. This means that all the sub-activities of this activity would be repeated x-times.

5.4.5 Process Decomposition Patterns

The constraints imposed on the process decomposition are specified as *ProcessDecompositionPattern* concepts which have the same principle structure as the *ActivitySpecificationPatterns* in section 5.4.3. The definition of the *ProcessDecompositionPatterns* is based in principle on the functional decomposition patterns proposed by Kitamura and Mizoguchi [61] and Kitamura and Mizoguchi [63]. Kitamura and Mizoguchi use a bottom-up approach in their work to derive higher level functional understanding of equipment functions based on a set of elementary functions and functional decomposition patterns. It seems plausible to use a similar mechanism to guide the decomposition in a top-down approach.

The *ProcessDecompositionPattern* concept defines the set of required sub-activity types, how they need to be temporally related, and what their <responsibleFor> attribute should refer to. The required <responsibleFor> attribute definition of the sub-*Activities* is associated to the attribute constraint of the

ProcessDecompositionPattern's *Activity* class. Furthermore, each *ProcessDecompositionPattern* can contain a description of the conditions under which it is applicable. This is required to guide decision making during the decomposition process. The conditions are defined based on either characteristics of the targeted state transition that need to be established or the agent that is responsible for the *Activity*. For example an *InsertionOperation* could occur either on a straight line or along some path depending on the specific characteristics of the *Liaisons* the *Operation* establishes.

The *ProcessDecompositionPatterns* are formally specified in First Order Logic using knowledge interchange format (KIF) notations (Genesereth and Fikes [42]). The use of First Order Logic enables the interpretation of the constraint both to check whether an instantiated process model is correct and to drive the decomposition process. The *ProcessDecompositionPattern* concept is defined as an extension of the *Specification* concept and has the additional attributes <range> and <statement> to define its First Order Logic constraints (see Figure 5.14). The <range> attribute is used to define the ranges of instance types that are used as part of the <statement> attribute to define the decomposition constraints.

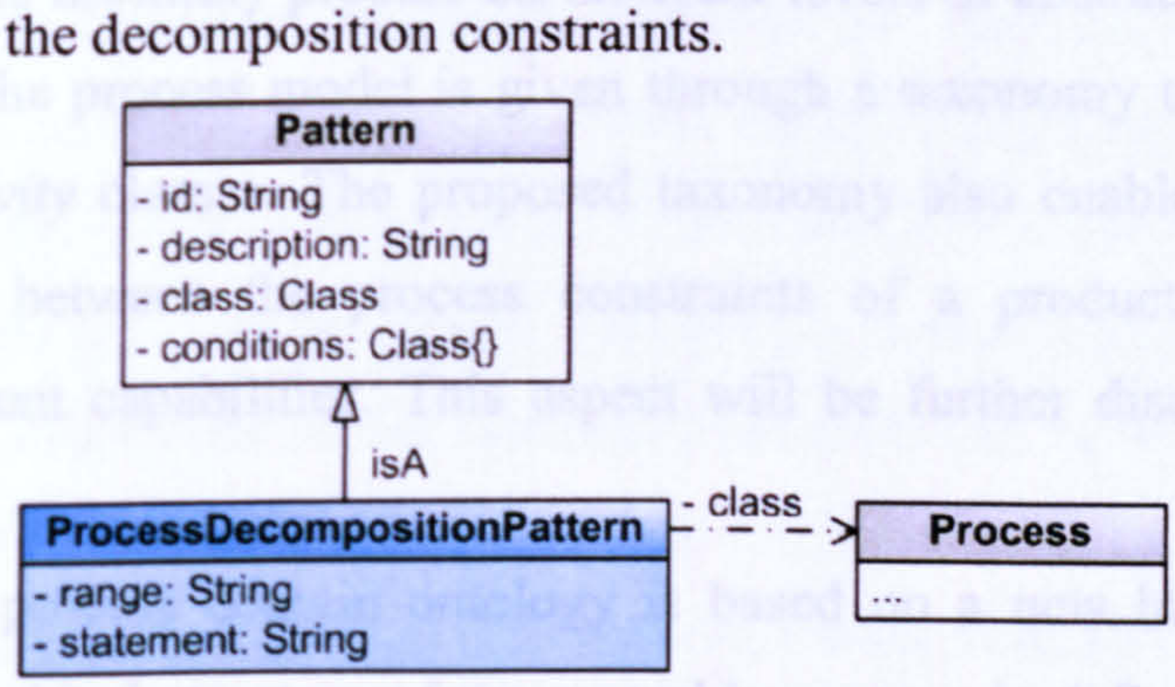


Figure 5.14 Process Decomposition Pattern Concept Definition

The requirement for the decomposition formalism to be able to express required and possible sub-activities is fulfilled by allowing the association of more than one *ProcessDecompositionPattern* to any type of *Process*. It is also possible to define the decomposition at different levels of abstraction since the decomposition patterns are directly associated to the activity taxonomy (see Figure 5.15).

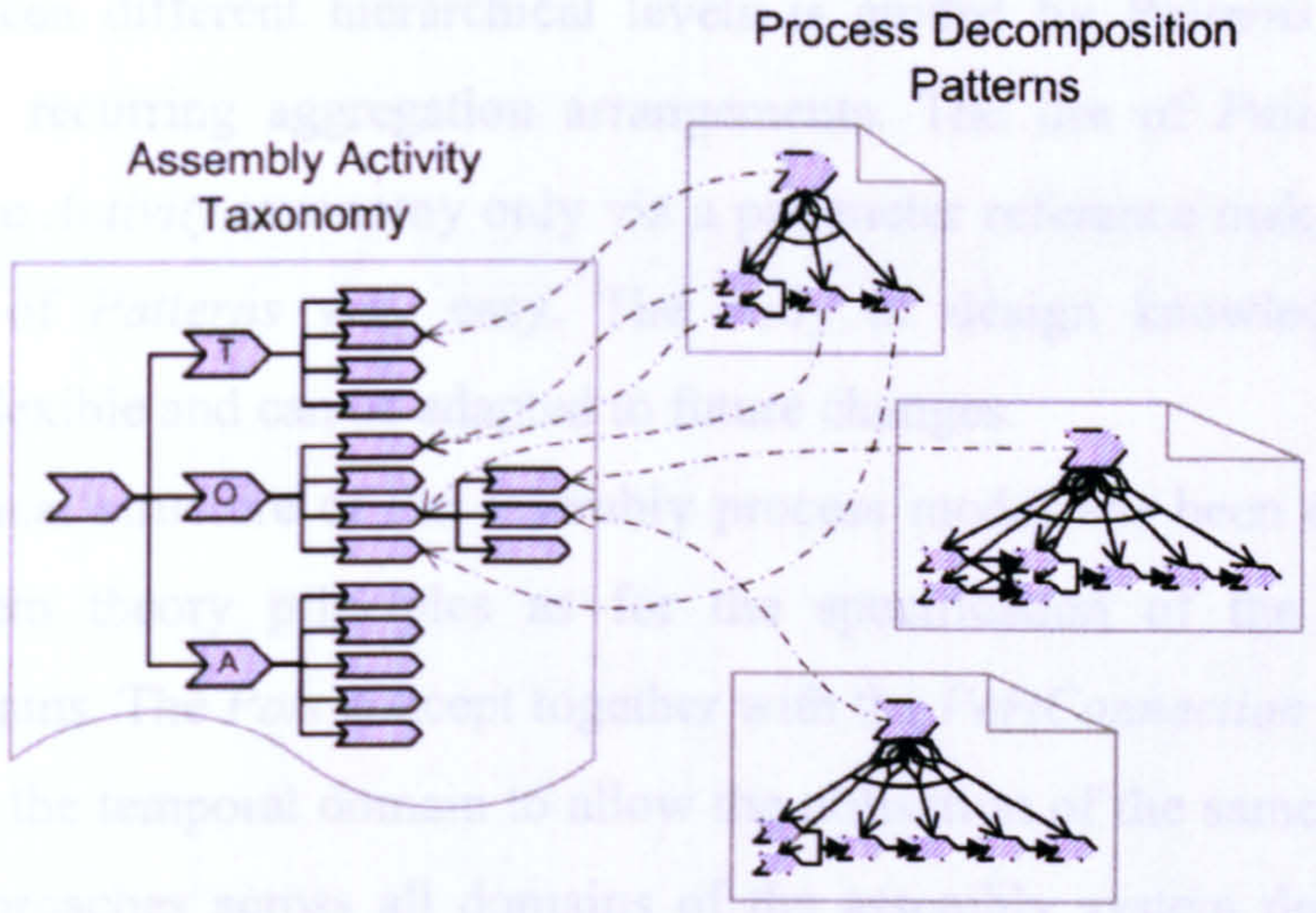


Figure 5.15 Principle Structure of Process Decomposition Pattern

5.5 Summary

In this chapter the assembly process domain conceptualisation of ONTOMAS has been defined and discussed. The central concept of the domain ontology is the *Activity* which is defined in the temporal space. The proposed formalisations allow a dynamic specification of the assembly process on different levels of abstraction. The heuristic interpretation of the process model is given through a taxonomy that clearly defines the different *Activity* classes. The proposed taxonomy also enables a clear purpose driven transition between the process constraints of a product and its required assembly equipment capabilities. This aspect will be further discussed in the next chapter.

The assembly process domain ontology is based on a new hierarchical process model. The hierarchical structure of the assembly process is defined using the *Task*, *Operation*, and *Action* concepts which reduce the inherent complexity of the assembly process definition. The clear hierarchical structure of the assembly process model combined with the proposed taxonomy of elementary *Activity* classes enables a clear association with the capabilities provided by different *Equipment* modules. This is very important for the engineering design process to capitalise on the higher level of standardisation inherent in a modular assembly system.

The proposed ONTOMAS framework also provides specific constraints that guide the specification of the assembly process. *Patterns* for the clear definition of the different *Activity* classes have been proposed to assist with the selection of appropriate *Activities* classes for given sets of requirements. The decomposition of assembly

processes between different hierarchical levels is guided by *Patterns* that capture recognised and recurring aggregation arrangements. The use of *Patterns* that are associated to the *Activity* taxonomy only via a parameter reference makes the adding and removing of *Patterns* very easy. The body of design knowledge therefore becomes very flexible and can be adapted to future changes.

The topological structure of the assembly process model has been defined using the same system theory principles as for the specification of the product and equipment domains. The *Port* concept together with the *PortConnection* concept have been applied to the temporal domain to allow the utilisation of the same fundamental engineering approaches across all domains of the assembly system design process. The application of the system theory principle also makes the definition of the temporal process structure much more dynamic since relative connections can be added and removed locally without influencing the rest of the definition. New process steps can therefore be integrated or removed quite easily.

The characteristics of the proposed assembly process conceptualisation are very beneficial for concurrent and iterative design approaches. Furthermore, the model can be used in temporal constraint engines to maintain the consistency of the process order. Also logical constraints between different enabling equipment behaviours can be derived from the process description which simplifies the definition of the equipment requirements. The proposed formalisms could ultimately be used to specify the control algorithms of the assembly system as can be seen in the work of Barata de Oliveira [6].

The next step of the knowledge transformation during assembly system design is the assembly equipment specification. The domain conceptualisation and its more specific relationship to the assembly process model have been defined and discussed in the next chapter.

6 Assembly Equipment Domain Ontology

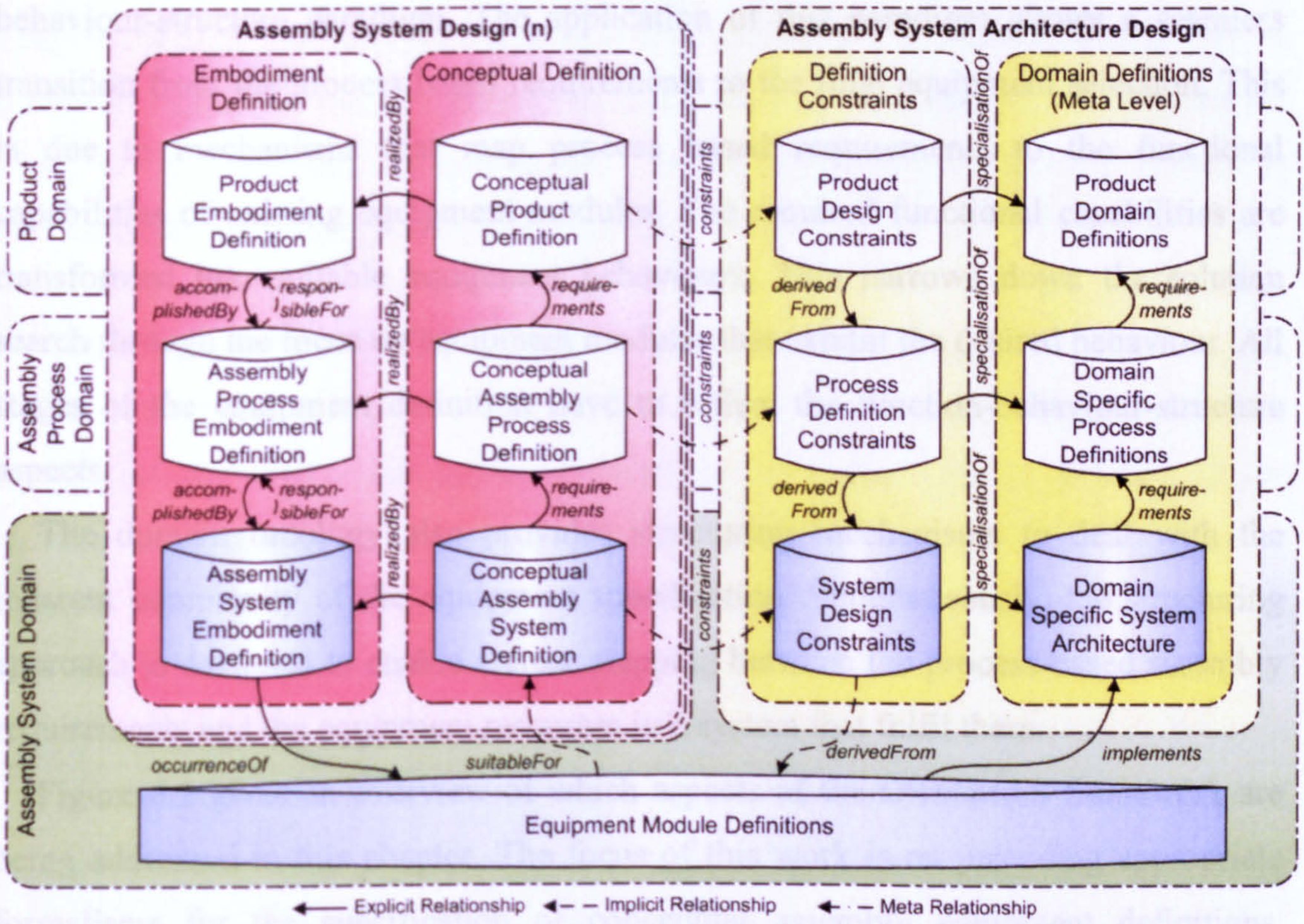


Figure 6.1 ONTOMAS Assembly Equipment Domain Ontology Overview

6.1 Introduction

The purpose of the propose assembly equipment domain ontology introduced in this chapter is to address the specific needs arising from the requirements driven design of modular assembly workstation solutions. The domain ontology needs to provide suitable conceptualisations that capture the required design knowledge and assist the decision-making process to achieve this.

The equipment domain conceptualisation has to address a number of different aspects to enable it to successfully support the design of modular assembly systems. Firstly, the ontology needs to provide formalisms for the specification of equipment requirements based on the assembly process requirements defined in the previous chapter. Secondly, the definition capability definition of existing equipment modules

needs to be addressed. Thirdly, the ontology needs to allow the integration of suitable equipment modules into assembly system solutions that can be verified against the original requirements. Finally, there need to be formalisms that guide and constrain both the design of individual assembly system solutions as well as the specification of new equipment modules.

The equipment domain conceptualisation of ONTOMAS is based on the function-behaviour-structure paradigm. The application of this paradigm allows a seamless transition from the process based requirements to the final equipment selection. This is due to mechanisms that map process based requirements to the functional capabilities of existing equipment modules. The required functional capabilities are transformed into suitable equipment behaviours. This narrows down the solution search through the focus on equipment modules that exhibit the desired behaviour. All stages of the equipment definition have to reflect the function-behaviour-structure aspects.

The domain ontology also provides structuring mechanisms to deal with the inherent complexity of the equipment specification. Simultaneously, the structuring approach is designed to enable a clear mapping between the process-based assembly requirements and the equipment resources in a system that fulfil them.

Figure 6.1 gives an overview of which aspects of the ONTOMAS framework are being addressed in this chapter. The focus of this work is on providing appropriate formalisms for the specification of conceptual assembly equipment definitions, assembly equipment embodiment definitions, system design constraints arising from the available set of equipment modules, and a domain specific system architecture that guides the equipment module specification. The system design constraints are important to make the design process more solution oriented and the system architecture to ensure interoperability between modules.

The chapter starts with discussing the informal terminology definitions chosen to describe the concepts of the assembly equipment domain ontology. This leads to the examination of their representational requirements. The next section describes how the requirements have been translated into a suitable assembly equipment domain conceptualisation. Finally, the chapter concludes with a summary of the proposed domain model.

6.2 Informal Assembly Equipment Domain Description

This section looks at the informal definition of the assembly domain specific terms that will be used throughout this work. The terms are being defined before the requirements for the domain model are being addressed because they will be used during the requirements specification to keep it more focused.

The terminology of the assembly equipment domain is, as in the previous two chapters, chosen to reflect the underlying principles of the proposed model as well as possible. Again, the terms used are not in themselves instrumental for the principle ideas of the proposed domain model but are nevertheless carefully chosen to maximise the explanatory value of the model.

6.2.1 Equipment Terminology

The most fundamental term within this domain is of course the term equipment itself. The term equipment has been chosen as the central term to describe all the entities that actively contribute towards the completion of an assembly process because it is the most general term to capture the character of these entities. The Merriam-Webster Online Dictionary defines equipment as:

> "the set of articles or physical resources serving to equip a person or thing as (1): the implement used in an operation or activity (2): all the fixed assets other than land and buildings of a business enterprise." (Merriam-Webster [81])

Based on the above definition equipment is both an asset of a company and used as an actor or implement in an activity. It therefore defines precisely all those entities that actively participate in a manufacturing or assembly process. The following definitions will be used throughout this work:

Definition: Manufacturing Equipment

"Manufacturing equipment describes all those physical entities that are used as actors to facilitate a manufacturing activity."

The term assembly equipment defines a subset of manufacturing:

Definition: Assembly Equipment

"Assembly equipment is all those equipment that is used as actors to facilitate an assembly activity."

Assembly equipment consists of a wide variety of different equipment types. It is advantageous for the effectiveness of the assembly system design process if there are defined associations between the concepts of the three domain models (see also discussion in chapter 5). The associations between the different types of equipment and the different types of activities, as shown in Figure 6.2, provide a good basis for the definition of the main types of equipment within the assembly domain.

The terms chosen for the different types of equipment are commonly used within the domain. Unfortunately there is no clear unambiguous definition for them across the whole domain. The terms system, cell workstation, unit, device, and element have been chosen to describe equipment with different levels of complexity. They are listed here in order of decreasing complexity. The complexity of the equipment is not directly referring to the structural or technical complexity of the equipment but rather to the complexity of the activities the equipment is design to perform.

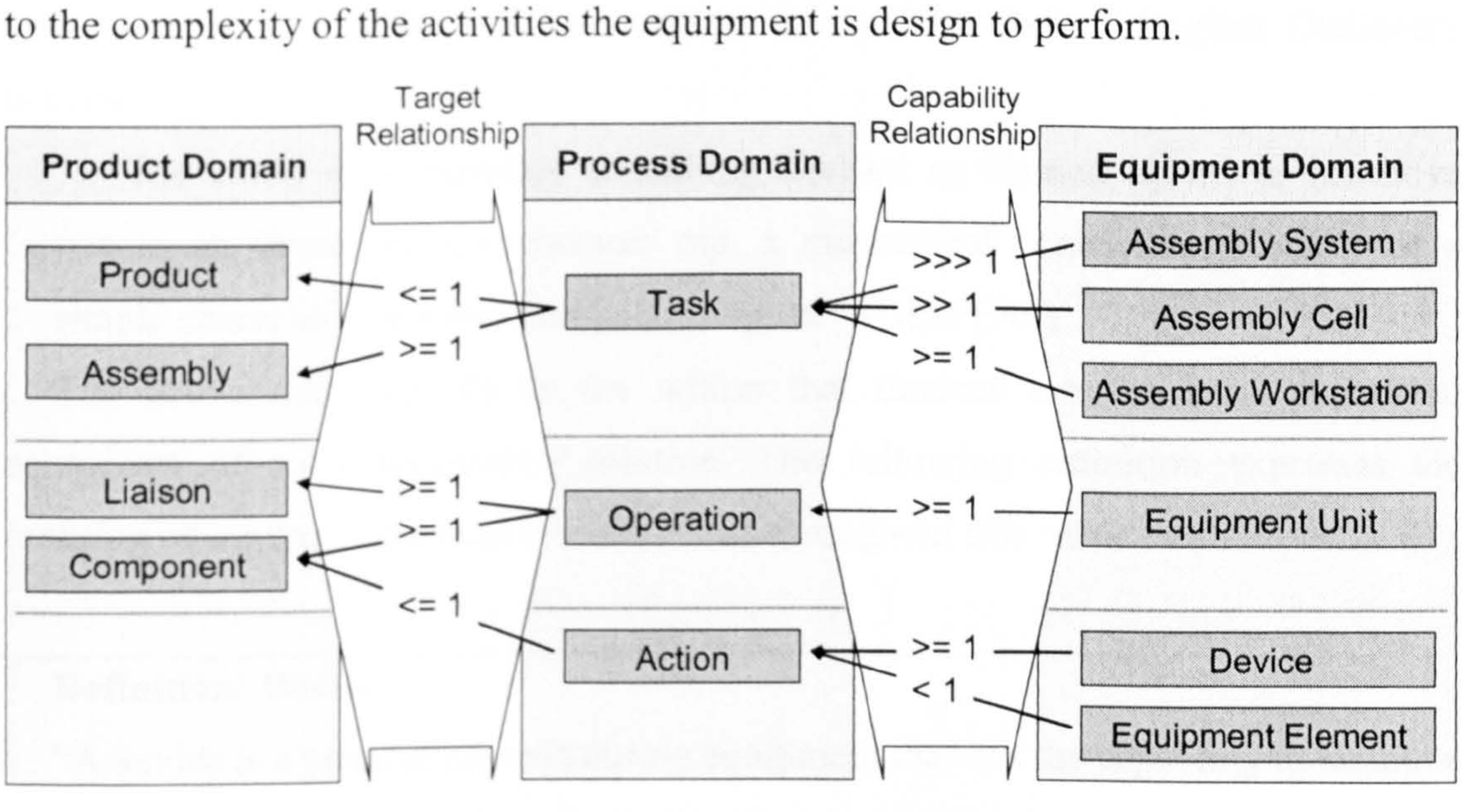


Figure 6.2 Associations between Domain Concept

The lowest complexity level of the equipment model is represented through the equipment elements. Equipment elements are the correspondent equipment concept to components in the product domain model. They do not fulfil any complete actions and only provide static or passive functions like support or constraining motions. The Oxford English Dictionary defines elements in general as:

"A component part of a complex whole." (OED [90])

This definition supports the decision to place equipment elements at the lowest level of the equipment definition model. The following definition will be used throughout this work:

Definition: Equipment Element

"An equipment element is a piece of manufacturing equipment that is part of another piece of equipment and does not directly facilitate the fulfilment of any activity."

Manufacturing equipment that can perform at least one complete action is defined as devices. This is based on the traditional use of the term device to describe most of the functional equipment used inside a manufacturing system. Devices are pieces of equipment that are ready bought in from companies specialising in their design and manufacture. Most of the equipment traditionally used to build manufacturing systems is called devices. This will become even clearer once the different types of devices have been classified (see section 6.4.5). The Oxford English Dictionary defines a device as:

"The result of contriving; something devised or framed by art or inventive power; an invention, contrivance; esp. a mechanical contrivance (usually of a simple character) for some particular purpose." (OED [90])

The above definition backs the notion that devices are the basic functional equipment of a manufacturing solution. The following definition expresses the meaning of the term device as it will be used throughout this work:

Definition: Device

"A device is a piece of manufacturing equipment that has the capability to facilitate at least one action towards the completion of a manufacturing process."

Whereas the two definitions above focused on somewhat elementary equipment entities, the following concepts focus more on different types of equipment configurations. Again the distinction between the different types is mainly based on their functional purpose. Traditionally manufacturing system and assembly systems in

particular are structured using the system, cell, and workstation concepts. They are normally associated with the fulfilment of one or more tasks.

The equipment unit concept has been introduced to bridge the gap between the definition of devices that are responsible for actions and the higher level equipment configurations that are responsible for tasks. The equipment unit definition is linked to the facilitation of operations on the process side. The term unit was chosen based on the notion that it represents sets of entities that can be combined with other units to form higher level systems:

> "A piece of furniture or equipment which may be fitted with other pieces to form a larger system, or which is itself composed of smaller complementary parts." (OED [90])
>
> "A piece or complex of apparatus serving to perform one particular function." (Merriam-Webster [81])

These two definitions of the term unit give justification to its intended use in the way described above. An equipment unit is for the purpose of this work defined as:

Definition: Equipment Unit

"An equipment unit is a set of manufacturing equipment that has the combined capability to facilitate at least one operation towards the completion of a manufacturing process."

The next higher levels are defined by the term workstation, cell, and system. They all define the fulfilment of tasks at increasing level of complexity. Their relationship to the product model has been added into their definition to create a clearer distinction between them. The workstation concept is on the lowest level of them. The term is defined by the Oxford English Dictionary as:

> "A location at which one stage in the manufacturing or assembly of a product is carried out before it is moved to the next stage." (OED [90])

This definition places the workstation clearly at the lowest level of the tasks fulfilling equipment configurations. It is linked to the fulfilment of at least one task and is therefore linked to the establishment of at least one liaison between two components. This definition is also creating the association with the notion of

something being stationary. This implies how the manufacturing task is taking place inside a station. Specifically for the assembly domain a workstation is defined as:

Definition: Assembly Workstation

"An assembly workstation is a set of manufacturing equipment that has the combined capability to perform at least one assembly task towards the completion of at least one assembly liaison."

The term assembly cell is normally associated with the complete assembly of at least one subassembly of a product. In this sense a cell is a set of workstations and other equipment required to completely assemble a subassembly. At the same time a cell is a subset of a system as will become clear a little later. This definition fits with the Oxford English Dictionary's definition of the term cell:

"[A cell is] one of the compartments into which anything is divided." (OED [90])

An assembly cell is therefore defined as:

Definition: Assembly Cell

"An assembly cell is a set of manufacturing equipment that has the combined capability to perform all required assembly tasks towards the completion of a subassembly."

The system defines the highest level of a manufacturing solution. The term system in this instance is not to be confused with its counterpart in system theory. All the above terms represent pieces of equipment that are treated as systems from the system theory viewpoint (see also section 6.4). The term assembly system for the classification of equipment is generally used to define the whole set of manufacturing equipment required to complete the assembly of a product. This description is backed by the Oxford English Dictionary's definition of the term system:

"[A system is] a set or assemblage of things connected, associated, or interdependent, so as to form a complex unity; a whole composed of parts in orderly arrangement according to some scheme or plan; rarely applied to a simple or small assemblage of things." (OED [90])

Based on the above argumentation an assembly system has been defined as:

Definition: Assembly System

"An assembly system is a set of manufacturing equipment that has the combined capability to perform all required assembly task towards the completion of a whole product."

6.2.2 Modular System Terminology

Additionally to the directly equipment related terms explored and described above there are some other concepts that are important for the equipment domain model. These include specifically terms that define how the equipment is designed and structured. The principle of modularity is one of the cornerstones that motivate this work. Hence the terms used related to modularity are being defined here. Furthermore, the proposed approach is relying on the existence of an architecture that prescribes how an assembly system has to be configured for a specific product domain. The term architecture is also clarified here.

The central terms of modularity are the modules and the interfaces between the modules (see chapter 2.2). A module is a specific type of entity. In this case the definition of a module is focused on the equipment domain. Equipment modules are highly standardised both in their function and in their interfaces through which they interact with their environment. This definition is supported by the Oxford English Dictionary:

> "[a module is] any of a series of independent units or parts of a more complex structure, product to a standard design in order to facilitate assembly and allow mass production." (OED [90])

The most important aspect of the module definition is its independence. That means a module needs to be decoupled from its environment and only allow an interaction through predefined interfaces. The definition also indicates one of the reasons for modularisation which is economies of scale from increased standardisation. An equipment module is defined as:

Definition: Equipment Module

"An equipment module is an independent piece of equipment with standardised functions and interactions with its environment."

The term interface is used to describe the interaction between two systems. These could for example be two connected modules or a module and its environment. The interface stands both for the definition of how the connection has to take place and what can cross the system boundary through the interface. The Oxford English Dictionary's definition of the term interface is:

> "A means or place of interaction between two systems, organizations, etc.; a meeting-point or common ground between two parties, systems, or disciplines; also, interaction, liaison, dialogue." (OED [90])

In order for two systems to fit together they need to implement the same interface. A degree of standardisation of interfaces is therefore desirable to support the idea of a modular system. The term interface will be used throughout this work as:

Definition: Interface

"An interface defines the place and way of interaction between two systems."

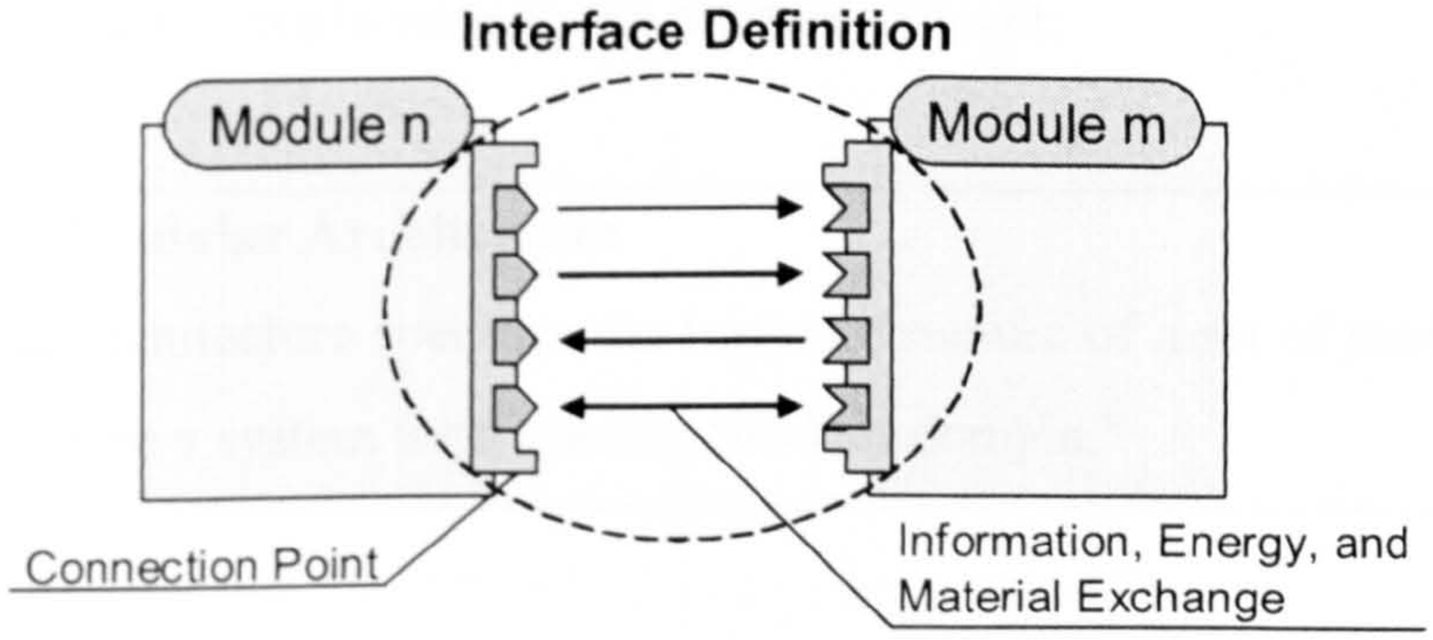


Figure 6.3 Interface Description

Figure 6.3 shows the principle structure an interface definition needs to describe. Besides the definition of how two modules connect using the same interface, the description also needs to contain a definition of the connection point and of the interaction that can occur through the interface. The term port will be used to describe the connection point of an interface.

Definition: Port

"A port specifies the point of interaction between two or more systems."

The term channel will be used to describe the interactions across the system boundary defined by the interface.

Definition: Channel

"A channel describes an interaction between two connected systems."

The set of different types of modules from which a specific system can be built is often called an architecture or modular system (Pahl and Beitz [94]). The term architecture will be used throughout this work to denote a set of module types and their physical and logical relationship constraints. The use of the term architecture is partially inspired from the computer domain. The Oxford English Dictionary defines it as:

"[An architecture is] the conceptual structure and overall logical organisation of a computer or computer-based system from the point of view of its use or design; a particular realisation of this." (OED [90])

The following definition of modular architecture can be made if the specialisation on computer system is replaced with the module concept:

Definition: Modular Architecture

"A modular architecture specifies the logical structure of a set of modules that can be use to configure a system for a specific product domain."

6.3 Equipment Domain Model Requirements

This section defines the requirements for the assembly equipment domain model. The purpose of the model is to support all the necessary activities during the embodiment design of assembly systems. These are the selection of suitable equipment modules, the configuration and reconfiguration of the selected equipment modules into suitable assembly system solutions, and the evaluation of alternative

configurations to verify that all requirements have been fulfilled and to select the most suitable solution.

Additionally the equipment domain model has to provide the modelling capabilities to define the conceptual aspects of the equipment design in order to capture the different configuration alternatives. The conceptual design is the transitional stage between the assembly process specification and the equipment selection and configuration. Consequently the model needs to allow the grouping of activities and their mapping to types of equipment (see chapter 3.5).

The assembly equipment domain model has to be defined in such way as to provide all the necessary domain knowledge to support the above activities in the most effective manner while at the same time being open enough to allow adaptation for future uses. The domain model therefore has to address the needs of the conceptual as well as the embodiment design and allow a seamless transition between the two. The equipment domain model needs to provide a mechanism to deal with highly complex models on both the conceptual and embodiment side in the same manner as in the product and assembly process domain.

In the following subsections the specific requirements for the conceptual and embodiment specification models will be discussed in more detail.

6.3.1 Conceptual Model Requirements

The conceptual design definition is both a qualitative behavioural view of the assembly process and an abstraction of the assembly equipment. It therefore needs to represent a clear link to the assembly process model and to the type of equipment it is an abstraction of. The structure of the conceptual equipment model needs to have distinct levels of hierarchy that reflect the hierarchy of the embodiment structure. An equipment concept needs to define the same decomposition into higher granularity equipment concepts as will be required from the equipment configuration. Furthermore the model needs to provide a formalism to define alternative variants of how the assembly process is being conceptualised for the equipment configuration. This is important to capture and understand the different choices during the system design. Not all of them need to be analysed to their full level of detail, but at least they should remain in the model to be detailed at a later point if required.

Each type of equipment concept needs to have a defined set of capabilities that define which assembly activities can be grouped to which equipment concept. The

representation of the capabilities needs to be in such manner as to allow an easy matching between the required activities on the process side and the existing functional capabilities on the equipment side.

The conceptual model needs also to represent the relationships between the equipment concepts on each level of hierarchy. The relationships need to define how the transformation of objects is facilitated between the different equipment concepts. The transformation itself takes place through the functional capabilities of the equipment. The relationships need to define how the objects that are being transformed move between the equipment concepts. The most critical definition is of course the transformation of the objects contained in the product model and how they come together.

Additionally to the structural definition of the model there is also a need for a mechanism to define the information required to evaluate the likely performance of the conceptual system. The main evaluation criteria are cost and time. This can be further split up into investment cost and operating cost on the cost estimation side and cycle time as well as system bottlenecks on the system balancing side. It is common practice to use discrete even simulation based software tools to determine the operational characteristics of a system. The definition of the equipment concept model should therefore provide the necessary information to be used in such software tools. The model needs to be structured in a way that allows a more accurate evaluation of the conceptual system with increasing specification accuracy of its model.

6.3.2 Embodiment Model Requirements

The role of the embodiment model is to support the selection, configuration, and evaluation of assembly equipment. The selection of assembly equipment is essentially a matching of the required assembly activities and the capabilities of existing equipment. On the lowest level, elementary assembly activities need to be compared with the capabilities of equipment at the lowest level. Hence, there needs to be a definable relationship between the activity definition and the equipment capability specification. Ideally there should be a one to one relationship between the two. This would not only make the matching much easier but is also fulfilling the requirement suggested by Vos [133] for equipment modules to be defined through their process scope. Additionally the equipment capabilities need to be defined in a way that they can be synthesised into higher level capabilities when two or more pieces of

equipment are combined into one functional configuration. This is important to understand if the combined capabilities still fulfil the process requirements at a higher level of abstraction. Particularly, since there are temporal constraints involved in the definition of the assembly process requirements and capability for co-ordinating the activity, related capabilities needs to be understood.

The configuration of lower level equipment into higher level more complex functional structures requires a clear definition of the connectivity constraints between the individual pieces of equipment. Consequently, the equipment embodiment model needs to provide a formalism to specify these constraints in a way that can easily be used during the configuration process. The connection specification also needs to define how the functions of the different equipment entities will be related to each other if they get connected.

Furthermore, the configuration of equipment requires the spatial definition of the equipment and the resulting configuration. The geometry of the equipment needs to be known and the connectivity constraints need to be defined relative to it. The geometry needs to be defined in the same way as in the product model (see chapter 4). The geometry is not only important for the static definition of the equipment configuration but also during the later behaviour simulation of the configuration. This is important to detect collisions between different pieces of equipment during the execution of the required assembly activities.

Finally, the evaluation of different equipment configurations requires a clear definition of how the equipment behaves under manufacturing conditions. This is required to evaluate the cycle time of the configuration and check for bottlenecks. A behavioural definition of the equipment can also be used to compare the dynamic properties of different equipment configurations and determine their accuracy. The dynamic assessment of the equipment configuration is outside the scope of this work, but it is important to define the model in such manner as to allow a later integration of dynamic models.

6.3.3 System Architecture

The purpose of an architecture definition is to limit the number of possible configurations to a set of more likely ones within a specific product domain. At the same time the architecture should not be so restrictive that it will rule out any kind of novel configurations that were not anticipated during the definition of the architecture.

This is particularly true if new equipment entities are being made available whose application is not well understood. The degree to which an architecture should prescribe the solutions depends on how well an application domain is understood. This work is focused on modular equipment solutions and it can therefore be assumed that the domain for which such solutions will be applicable is quite well understood, at least to the degree as to allow modularisation of equipment.

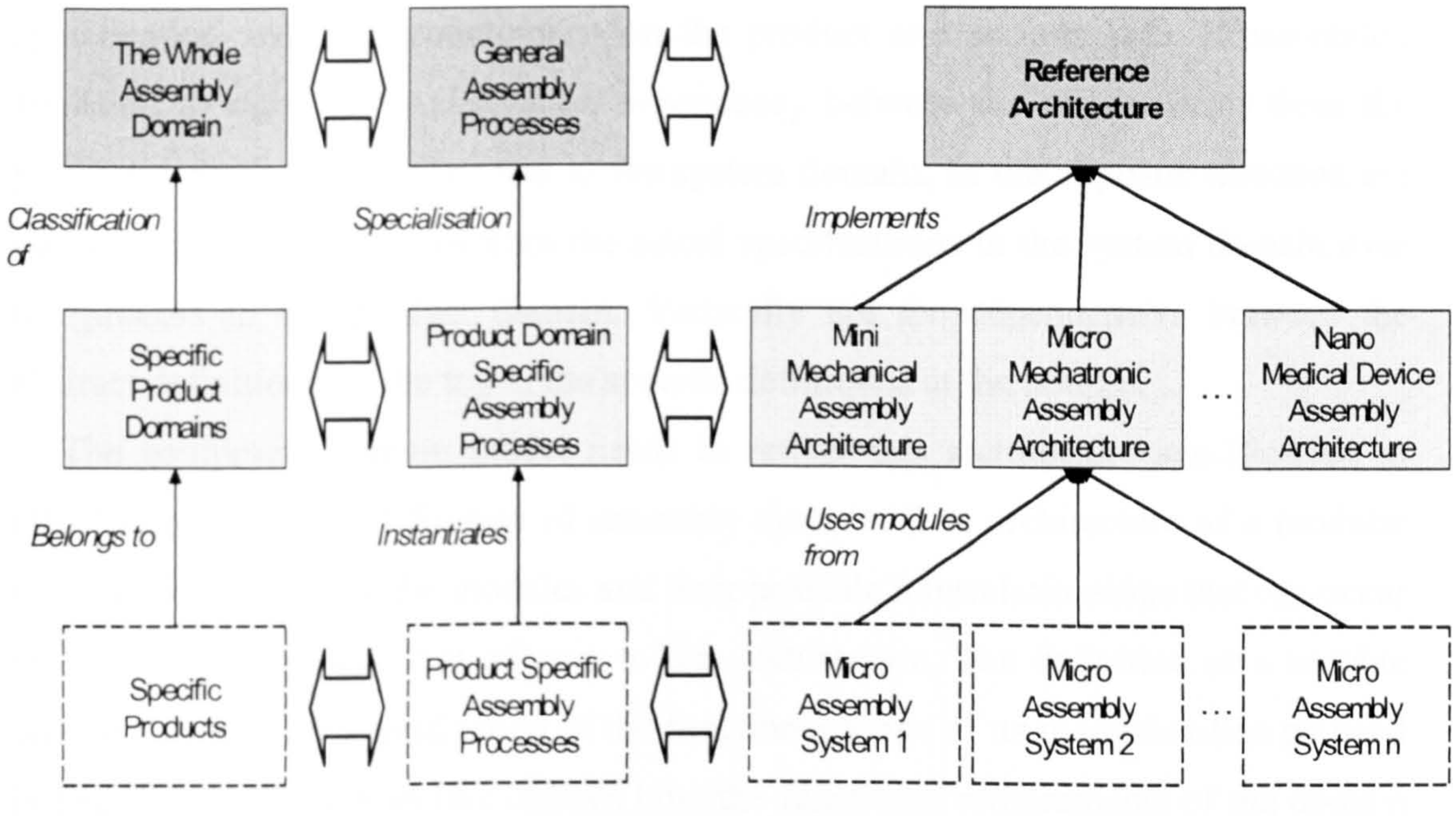


Figure 6.4 Architecture specification

An architecture can be defined on three levels of abstraction (see Figure 6.4). The most abstract is the definition of how an architecture needs to be defined. This definition is normally called a reference architecture which is essentially the fundamental concepts and a set of methods for the specification of new architectures (Zwegers [146]). The specification methods for an architecture are outside the scope of this work. The core ontology defined in chapter 3 does, however, reflect the fundamental concepts of the reference architecture.

On the second level are the domain specific architecture specifications. They define the modular system structure for the specific process requirements of a particular product domain like, for example, the domain of micro mechatronic products. Zwegers [146] distinguishes between a reference model and an architecture. He defines a reference model as describing "the generic manner to organise and integrate system components" and an architecture as describing "the manner in which the components of a specific system are organised and integrated". In this work the term architecture is used to describe what Zwegers calls a reference model. It gives

the definition of modules and their interconnections that can be used to create specific system instantiations.

The actual system instantiations are on the lowest level of abstraction. They deliver the assembly processes for a specific product and need to adhere to a specific modular system architecture.

Figure 6.4 shows the relations between the different levels of an architectural specification and their counterparts on the product and process side. Horizontally, from left to right, is the derivation dependency between the requirements from the product domain over the process to the system domain. In the opposite direction are the constraint dependencies from the actual specifications in the system domain over the process to the product domain. Vertically are the dependencies between the abstract definitions on the top to the specific definitions at the bottom.

The equipment domain model needs to reflect this architecture specification to effectively guide the definition of assembly systems. The architecture of a modular system needs to define the modules and their possible interrelationships that can occur in any system instance that adheres to this architecture. The definition of a suitable architecture and the specification of the functional scope of its modules often go hand in hand since the architecture defines how the functional requirements of the domain are covered by its set of modules and their interface constraints (Pahl and Beitz [94] and Zwegers [146]). The functional scope of modules for manufacturing systems should be defined based on the required process capabilities of a manufacturing domain to achieve a high level of module interchangeability which is important for adaptable and reconfigurable systems (Vos [133]). This requires that the functional capabilities of a module need to reflect clearly how the module contributes to the implementation of assembly processes.

The interrelationships between the modules defined in the architecture should cover two aspects: how two modules can be physically connected and the logical constraints imposed to achieve aggregate capabilities through the connection of different modules.

The physical connection constraints of an architecture need to define clearly how two modules can be connected to each other and what the behavioural implications of the connection has to be. That means that two modules have to have fitting connection points and their connection results in specific exchange of signals and material flows.

The logical constraints of the architecture should define some of the constraints that are imposed on the connections between different types of modules to achieve specific synthesised capabilities. For example, if a module needs to be configured that can deliver the insertion of a component then this would imply the connection of a module that can move in the desired fashion and a module that can hold the component.

6.4 Formal Modular Assembly Equipment Domain Model

In this section the formal modular assembly equipment model is defined and discussed. In the course of this section, the structure of the equipment model will be described and it will be shown how the model meets the modelling requirements defined in section 6.3. The main purpose is of course to provide an effective modelling framework for the selection and configuration of modular assembly equipment.

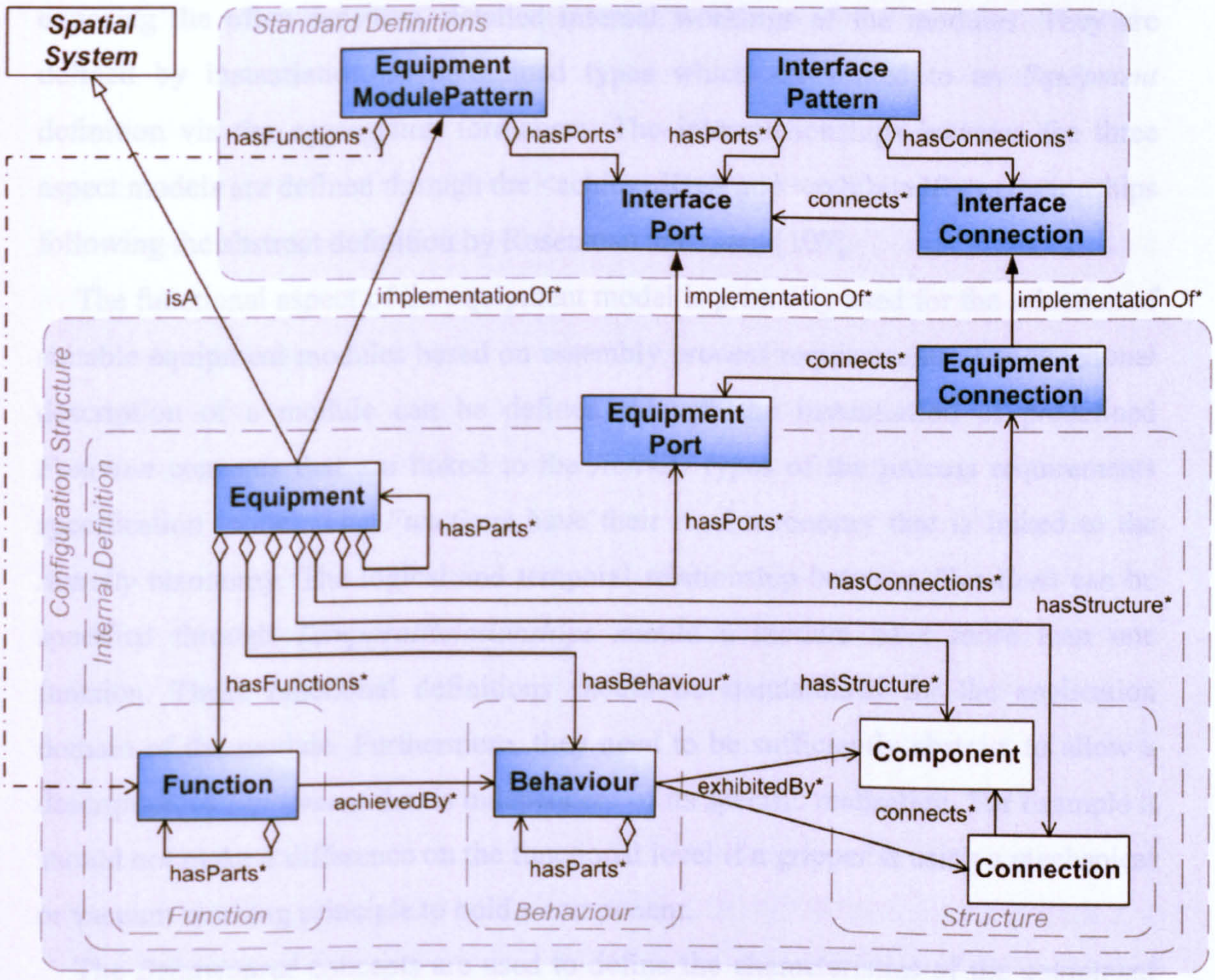


Figure 6.5 Assembly Equipment Domain Conceptualisation Overview

The formal model for the assembly workstation configuration method needs to cover three aspects: the conceptualisation of the needed equipment, the embodiment of the actual equipment and how it is being connected, and the specification of an architecture to guide the configuration process. Figure 6.5 shows the relationships between the main concepts used to describe the above aspects of the equipment domain model. The *Equipment* is the central concept of the domain conceptualisation.

The *Equipment* concept provides the formalism to view the equipment as a closed system that allows only interactions via defined *EquipmentPorts*. This is an essential requirement for a model that will be used to define modular solutions, reflecting the assumption that each module is designed and optimised for a specific function. This allows and requires detailed definitions to be hidden to reduce the design effort on module configuration level.

The function, behaviour, and structural aspect models only provide the right level of detail to enable effective system configuration and reconfiguration without exposing the often sensitive, detailed internal workings of the modules. They are defined by instantiation of predefined types which are linked to an *Equipment* definition via the aggregation formalism. The interrelationships between the three aspect models are defined through the <achievedBy> and <exhibitedBy> relationships following the abstract definition by Rosenman and Gero [107].

The functional aspect of the equipment model is primarily used for the selection of suitable equipment modules based on assembly process requirements. The functional description of a module can be defined through the instantiation of predefined *Function* concepts that are linked to the *Activity* types of the process requirements specification model. The *Functions* have their own taxonomy that is linked to the *Activity* taxonomy. The logical and temporal relationship between *Functions* can be specified through *TemporalRelationships* should a module have more than one function. These functional definitions should be standardised for the application domain of the module. Furthermore, they need to be sufficiently abstract to allow a description of *Equipment* that is independent of its specific realisation. For example it should not make a difference on the functional level if a gripper is using a mechanical or vacuum working principle to hold a component.

The *Behavioural* concepts are used to define the characteristics of the equipment more closely related to its working principle. The model is using information, energy, and material transformations and flows to provide a basic framework for different

behavioural aspects to be defined. In the assembly domain these predominantly include kinematic, dynamic, logistical, and component holding behaviours.

The structural aspect model provides two formalisms to define the internal geometric-spatial structure of the equipment entities and to define the *Connection* relationships between sets of equipment entities (layout). The *Component* and *Connection* concepts are used to define the geometric-spatial structure of the equipment at a level of abstraction that corresponds to the *Behaviour* definition. For example for the kinematic/dynamic representation of a robot it is sufficient only to define its links and joints without having to include all the details on its individual part composition. This philosophy has been used and proven itself in many robotic simulation environments. The arrangement of different equipment entities can be defined by linking *EquipmentPorts* with *EquipmentConnections* to each other. Both the *EquipmentPort* and the *EquipmentConnection* concepts describe not only the physical connection, but also the functional and behavioural interrelationships between two connected *Equipment* instances. This allows manufacturing systems to be defined by only connecting and configuring of their *Ports*. The definition of equipment configurations can be hierarchically structured with the <hasParts> aggregation relationship between *Equipment* instances. A number of distinct hierarchical levels have been defined for the assembly system design domain. These are logically linked to the hierarchical structure of its assembly functions to allow a structured validation of complex arrangements at different levels of detail as is commonly used in system engineering approaches (Stevens, et al. [118]).

All the definitions explained thus far only address the modelling of equipment in general and do not specifically reflect the implications of modular equipment frameworks beyond some structuring choices. This has been intentional to show that the proposed model is not restricted to the definition of modular equipment but is rather complemented by it. The modular paradigm is essentially a set of guidelines and conventions that are imposed on a specific domain to enable independently defined entities to fit together into a wider system. They are often results of observed recurring design patterns in a domain. This idea has been incorporated into the proposed assembly equipment domain ontology. A set of concepts represent the standards or patterns that apply for a given modular system architecture. These define different module types (*EquipmentModulePattern* concept) and the interfaces between them (*InterfacePattern* concept). The *EquipmentModulePattern* concept defines

which functions and which connection possibilities a piece of equipment needs to provide to qualify as a specific type of module. The *InterfacePattern* concept defines the characteristics two ports need to fulfil to allow their connection to each other. The definition of these standards or patterns can either be used to guide the design of modules or to recognise if an existing piece of equipment already adheres to a given equipment module definition. A piece of equipment could consequently be the implementation of more than one equipment module definition. This allows equipment entities to be used in either more than one location in one system architecture or in different system architectures. This makes the chosen approach very dynamic and closely reflecting the philosophy of modular assembly systems.

All concepts of the assembly equipment domain ontology are either directly or indirectly derived from the central *CLASS* concept. Consequently, they all inherit the three aspect definitions described in chapter 3.5. The aspect addresses the specification of the requirements for a specific concept class, available characteristics for the concept class, and the actual characteristics that an instance of the class has been set to fulfil the requirements.

The *Equipment* is the central concept of the assembly equipment domain ontology since it describes the characteristics of the equipment entities the assembly system will be configured from to carry out the assembly of the product. As a result, only the *Equipment* concept will be considered in more detail regarding its aspect definitions. The other concepts however do have aspects as well which is especially important for the *Function* and *Behaviour* definition.

The requirements aspect of the *Equipment* concept describes the manner in which the *ActivityRequirements* of the conceptual assembly process model need to be carried out. This definition provides the central input for the selection of the right assembly equipment (see chapter 7 for more details). The available *Equipment* characteristics are specified based on characteristics of the physical existing equipment entities. This is one of the most important aspects of the whole ONTOMAS framework since all the other constraints and guidelines are derived from this definition. The actual *Equipment* characteristics express the specific setup of the available equipment entities for a given set of requirements.

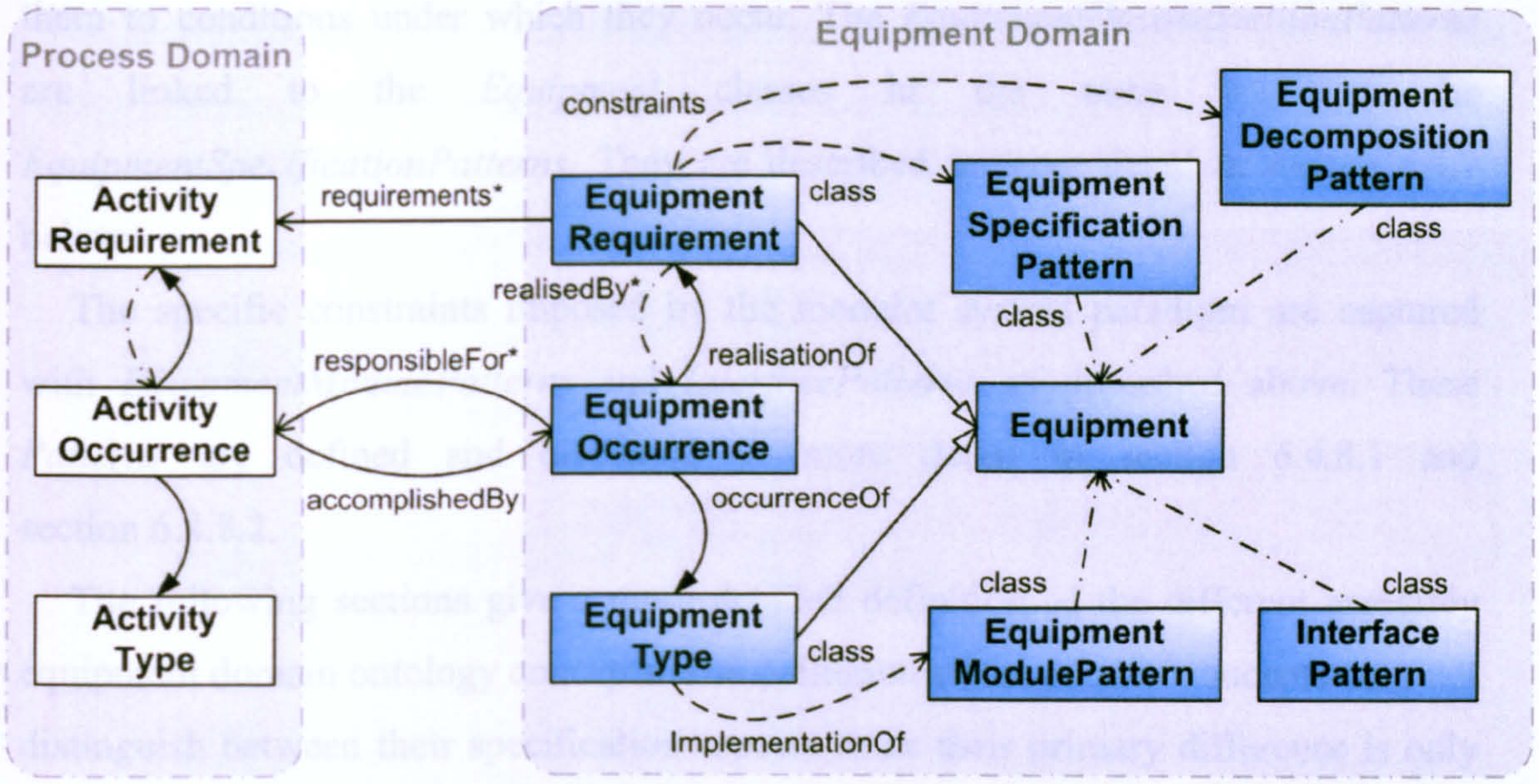


Figure 6.6 Relationships between the aspects of the main domain concepts

Figure 6.6 shows the aspect relationships between the central concepts of the product and assembly process models. The *EquipmentRequirement* concept defines the required characteristics for the *Equipment* through the <requirements> attribute. The concept allows the selection of an appropriate *Equipment* class for its specific requirements with the <class> attribute. The *EquipmentOccurrence* concept specifies the actual instance of the *Equipment* that realises the requirements specified by the *EquipmentRequirements* concept. The <responsibleFor> attribute associates the *EquipmentOccurrence* with the actual part of the assembly process model it is responsible for carrying out.

ONTOMAS provides the formalisms for an *Equipment* taxonomy. The taxonomy defines different classes of *Equipments* that could occur as part of an assembly system specification. The tree structure of the taxonomy fulfils the need of the engineering design process to support the abstraction and specialisation during the assembly system specification. The basis for the classification of in the taxonomy is captured by *EquipmentSpecificationPatterns*. They define the difference between alternative *Equipment* classes. A more detailed definition and discussion of the *EquipmentSpecificationPattern* can be found in section 6.4.5 below.

Another engineering process that takes place during the assembly system specification is the decomposition of higher level equipment requirements into more detailed lower level descriptions. This process can be assisted and constraint with *EquipmentDecompositionPatterns* that capture recurring aggregation patterns and link

them to conditions under which they occur. The *EquipmentDecompositionPatterns* are linked to the *Equipment* classes in the same way as the *EquipmentSpecificationPatterns*. They are described in more detail in section 6.4.7 below.

The specific constraints imposed by the modular system paradigm are captured with *EquipmentModulePatterns* and *InterfacePatterns* as described above. These *Patterns* are defined and discussed in more detail in section 6.4.8.1 and section 6.4.8.2.

The following sections give a more detailed definition of the different assembly equipment domain ontology concepts. The definition of the domain concepts does not distinguish between their specification aspects since their primary difference is only in their instantiation.

6.4.1 Equipment Concept Definition

Equipment entities are essentially the same as the *Products* discussed in chapter 4. They tend to be more complex than the *Products* that are being assembled, but this is not always the case. For example the assembly of cars requires much the same assembly equipment as the assembly of a toaster. The difference lies in principle only in the amount of equipment used and its size. The product in this case would be more complex than the equipment that is being used to assemble it. Furthermore, a piece of equipment for the designer and user of an assembly system is a product for its manufacturer. Hence the *Equipment* concept is defined as an extension of the *Product* concept (see Figure 6.7).

More importantly, since the *Product* concept is an extension of the *PhysicalSystem* concept, the *Equipment* concept inherits the <hasBehaviour>, <hasFunction>, and <hasStructure> attributes to define the equipment. Furthermore, since it is derived from the *System* concept its interactions with its environment can only take place through defined ports which are specified by the <hasPorts> attribute. All four of the above attributes are interpretations of the <hasParts> attribute. The <hasBehaviour>, <hasFunctions>, and <hasStructure> attributes do not really have to be defined as separate attributes since they could be derived with an algorithm. They have only been defined as separate attributes to emphasise the different roles of the <hasParts> attribute. The <hasPorts> attribute on the other hand needs to be defined separately even so, the *Port* instances it references are also included in the <hasParts> attribute.

The reason for that is that the <hasParts> attribute contains also the *Ports* of lower level *Equipment* entities which are not exposed at this level.

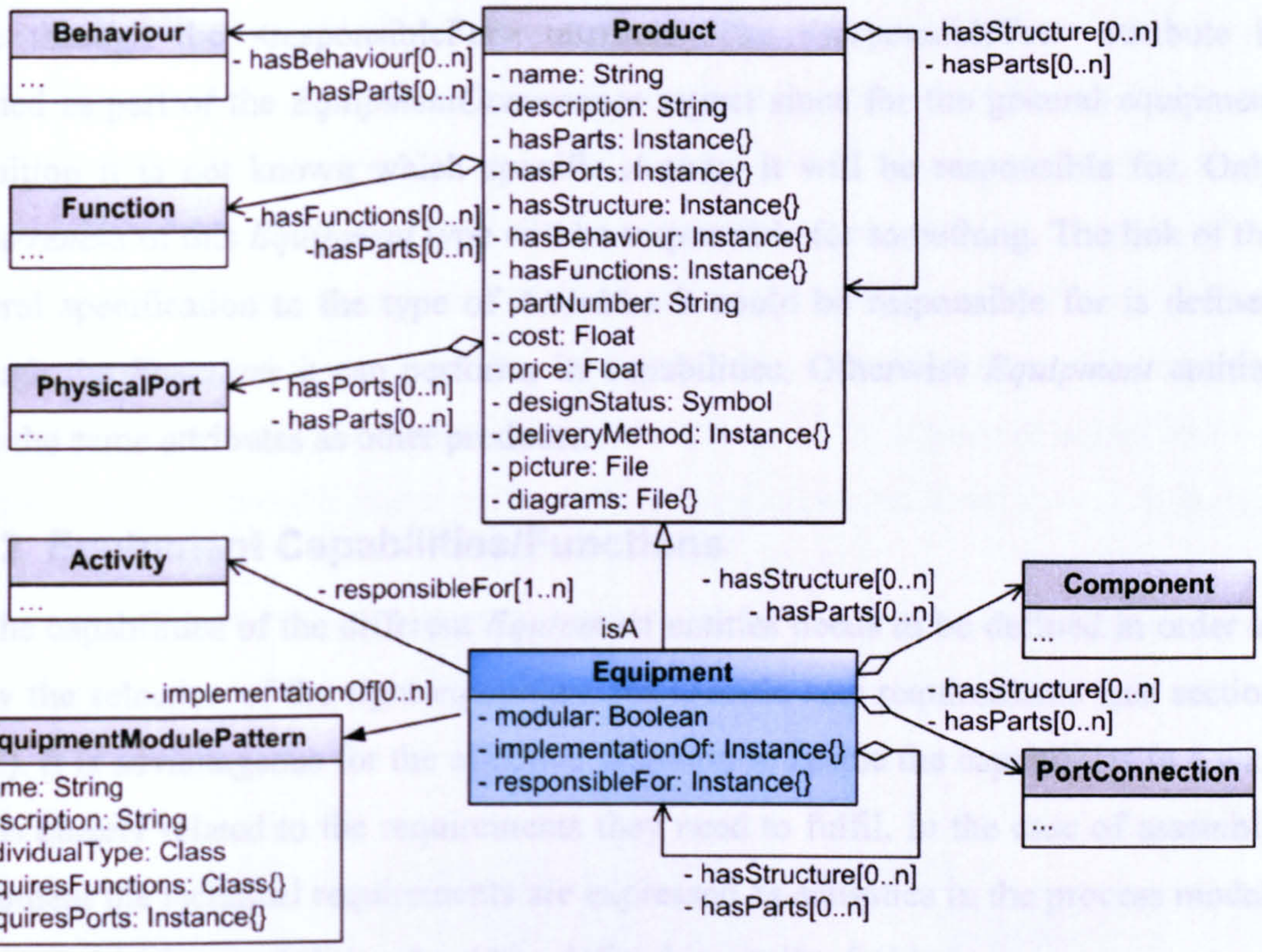


Figure 6.7 Equipment Concept Definition

Since modularisation of the physical hardware is one of the underlying paradigms of this work, each piece of *Equipment* needs to identify what type of *Equipment* module specification it adheres to and whether it has a modular structure. Its compliance to a module specification is defined through the <implementationOf> attribute. The *EquipmentModulePattern* concept defines which *Functions* the *Equipment* entity needs to satisfy and which *InterfacePorts* it needs to implement (see section 6.4.8.1 for a more detailed definition). The *EquipmentModulePattern* concept acts as a standard definition of the different types of modules. A piece of *Equipment* claims implementation conformity to a specific module specification by linking it to an *EquipmentModulePattern* definition. For the specification of it in the model, that means that its functional definition and ports need to match the *Functions* and *Ports* defined in the *EquipmentModulePattern*. The <modular> attribute defines whether or not a composite *Equipment* entity has a modular structure. It can either be true or false. An *Equipment* entity with a modular structure can only have *Equipment* entities that are modules as sub-entities on the next lower level of hierarchy.

Once an *Equipment* entity is selected to become part of a system configuration, it will get a specific part of the assembly process allocated as its responsibility. This is done through the <responsibleFor> attribute. The <responsibleFor> attribute is defined as part of the *EquipmentOccurrence* aspect since for the general equipment definition it is not known which specific *Activity* it will be responsible for. Only *Occurrences* of this *Equipment* type will be responsible for something. The link of the general specification to the type of *Activities* it could be responsible for is defined through the *Functions* it can perform, its capabilities. Otherwise *Equipment* entities have the same attributes as other products.

6.4.2 Equipment Capabilities/Functions

The capabilities of the different *Equipment* entities needs to be defined in order to allow the selection of the right equipment for specific sets requirements (see section 6.3.2). It is advantageous for the effective selection to define the capabilities in a way that is closely related to the requirements they need to fulfil. In the case of assembly equipment the technical requirements are expressed as activities in the process model. Consequently the capabilities should be defined in similar fashion.

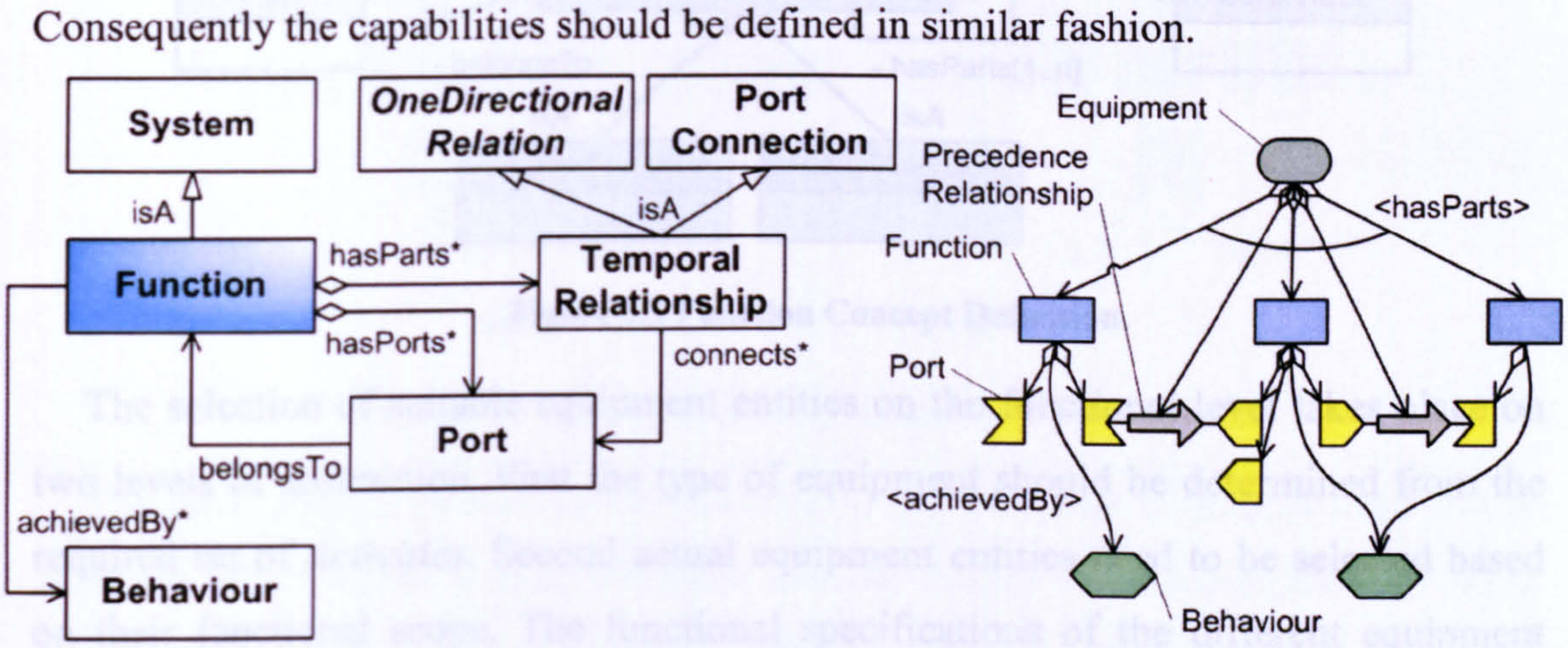


Figure 6.8 Function Definition Overview

Traditionally, the required capabilities of equipment are defined through functions (see also section 6.2.1). For this reason the *Function* concept (see Figure 6.8) has been used to define the high level capabilities of the different *Equipment* types and instances. If an *Equipment* entity has more than one *Function* there is often a precedence relationship between them. For example, a conveyor with a stop gate could be said to have two sequential transport functions. Hence its second transport function could only be used after the first one. The functional capability specification

is normally quite abstract and it is therefore linked to the behavioural model which is much more detailed and objective. This relationship is defined through the <achievedBy> attribute of the *Functions*.

The *Function* concept has been divided into *ActiveFunctions* and *PassiveFunctions* (see Figure 6.9). The *PassiveFunctions* define attributes of the equipment that cannot be controlled or activated from outside. A *PassiveFunction* could for example be "to support something" in the sense of a table or other structural element. These functions are also important but they do not directly relate to the requirements coming from the process specification. *ActiveFunctions* describe the activities a piece of equipment can perform. They are directly linked to the *Activity* they represent through type definition patterns.

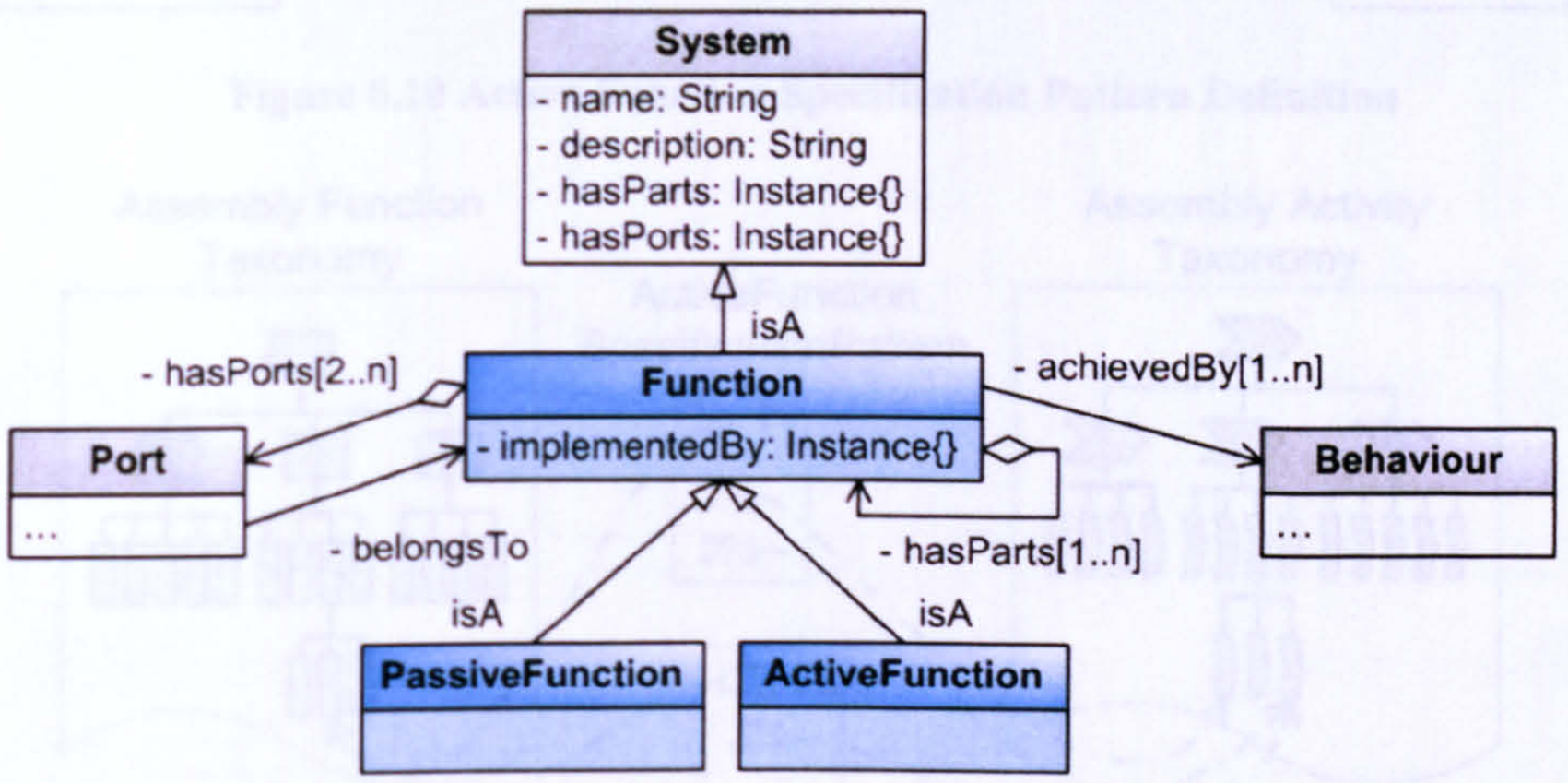


Figure 6.9 Function Concept Definition

The selection of suitable equipment entities on the functional level takes place on two levels of abstraction. First the type of equipment should be determined from the required set of *Activities*. Second actual equipment entities need to be selected based on their functional scope. The functional specifications of the different equipment types need to be compared with the required activity types during the selection of the right type of equipment (see chapter 7 for more details). This is a matching process that requires a defined relationship between the required activity types and the functional capability definition of the equipment entities. An ideal scenario for the matching would be that there was a one-to-one relationship between the functions defining the equipment capabilities and the activities defining the requirements.

It is not always possible or desirable, however, to define such a one-to-one relationship. In some instances it might be more desirable to have a different association, for example, to allow different domain specific terminologies for

Functions to be used. The *ActiveFunctionSpecificationPattern* concept has been defined to specify an AND/OR relationship between *ActiveFunction* types and Activity types (see Figure 6.10). The <class> attribute is related to the *ActiveFunction* types and defines the OR relationship. The <enables> attribute links to the *Activity* types and defines the AND relationship. Figure 6.11 shows an illustrative example of an *ActiveFunctionSpecificationPattern* definition.

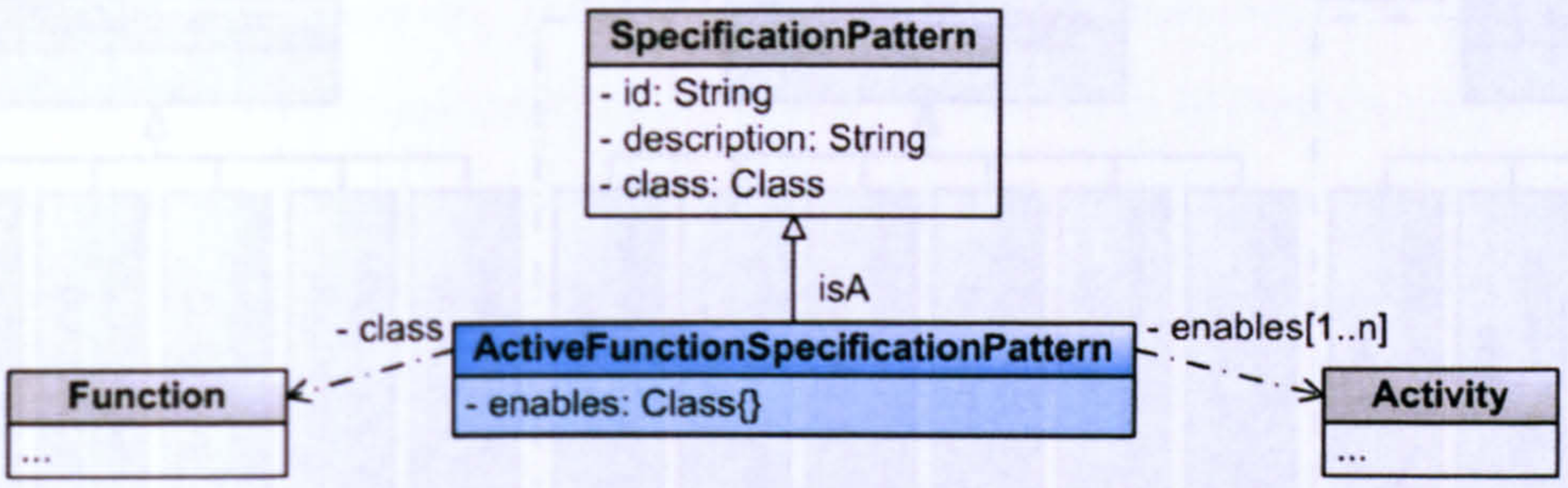


Figure 6.10 Active Function Specification Pattern Definition

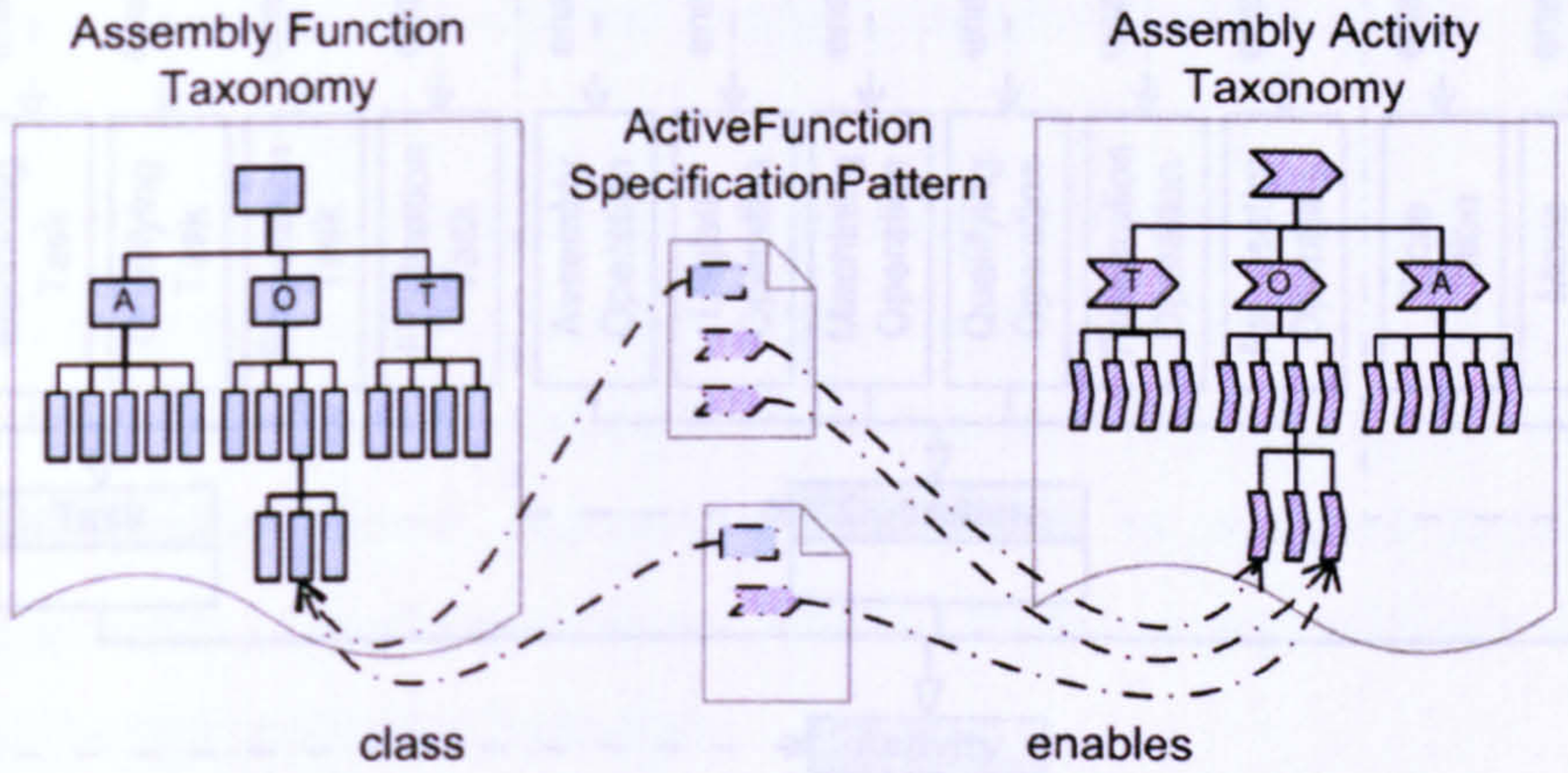


Figure 6.11 Active Function Specification Pattern Example

The classification of the *ActiveFunctions* in the *Function* taxonomy used in this work has been defined in such a manner that the *ActiveFunctionSpecificationPattern* establishes a one-to-one relationship between the *ActiveFunction* types and the *Activity* types in the assembly process taxonomy. Figure 6.12 shows the principle relationships that have been defined through *ActiveFunctionSpecificationPatterns* between the top level definitions of the *ActiveFunction* taxonomy and the *Activity* hierarchy. It can be argued that it would not be necessary to define a detailed *Function* taxonomy since it is literally the same as the *Activity* taxonomy; however, in order to be able to clearly define the functional constraints for the different types of *Equipment* and to allow a later integration of domain specific *Function* types, it is necessary to define the functional taxonomy separately.

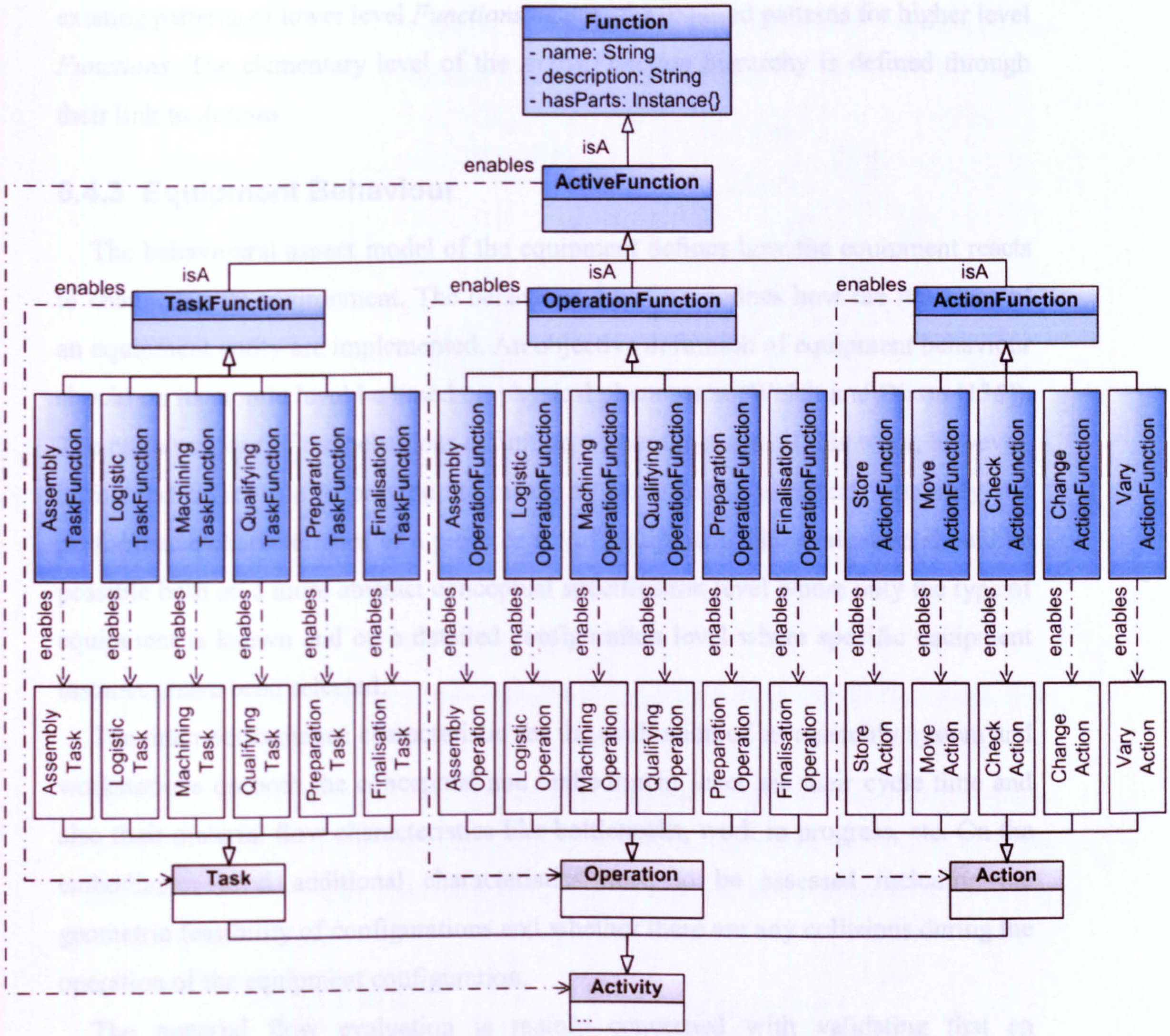


Figure 6.12 Active Function and Activity type relationships

During the selection of actual *Equipment* entities the required *Activities* can be compared with the *ActiveFunctions* implemented by the different *Equipment* entities. Once the *Equipment* has been chosen it will get the responsibility for the selected set of *Activities* assigned to it.

The synthesis of *Functions* into higher level *Functions* is defined in the same way as for all *System* concepts through the <hasParts> attribute. The definition of which lower level *Functions* enable specific higher level *Functions* is, in the case of the *ActiveFunctions,* based on the activity decomposition patterns that are also used to guide the decomposition of assembly processes (see chapter 5). They describe possible ways in which *Activities* can be broken down into more detailed lower level *Activities*. During the functional synthesis the reverse process can be used to match

existing patterns of lower level *Functions* against the required patterns for higher level *Functions*. The elementary level of the *ActiveFunction* hierarchy is defined through their link to *Actions*.

6.4.3 Equipment Behaviour

The behavioural aspect model of the equipment defines how the equipment reacts to changes in its environment. The behaviour therefore defines how the functions of an equipment entity are implemented. An objective definition of equipment behaviour should on its atomic level be based on physical phenomena (Welch and Dixon [135]). The main purpose of the behaviour definition within the scope of this work, however, is focused on its role to provide the means to evaluate the technical suitability and performance characteristics of a piece or set of equipment. The evaluation should be possible both on a more abstract conceptual specification level where only the type of equipment is known and on a detailed configuration level where specific equipment instances have been selected.

The main performance characteristic for the evaluation of an assembly system and workstations on both the conceptual and embodiment level are their cycle time and also their material flow characteristics like bottlenecks, work in progress, etc. On the embodiment level additional characteristics need to be assessed including the geometric feasibility of configurations and whether there are any collisions during the operation of the equipment configuration.

The material flow evaluation is mainly concerned with validating that an equipment configuration meets the requirements imposed by the temporal order of the assembly process it needs to achieve. For example if task 1 has to be followed by task 2 then the workstation responsible for task 1 has to have a material flow to the workstation responsible for task 2. The assessment of the cycle time on the conceptual level does not need to give exact times but should establish whether or not the desired cycle times are possible to achieve with the given equipment configuration. Hence each type of equipment needs to have an average cycle time for specific activities associated to it.

The *Behaviour* concept is very closely related to both the structure and the *Functions* of the *Equipment* (see 6.3). The *Behaviour* concept needs to be part of the *Equipment* definition. This is modelled through the <hasParts> and <hasBehaviour> attribute of the *Equipment* concept (see Figure 6.7). Additionally, the relationships

between the *Behaviours* of connected *Equipment* need to be defined. This is done as part of the *PortConnection* definition of *Equipment* concepts.

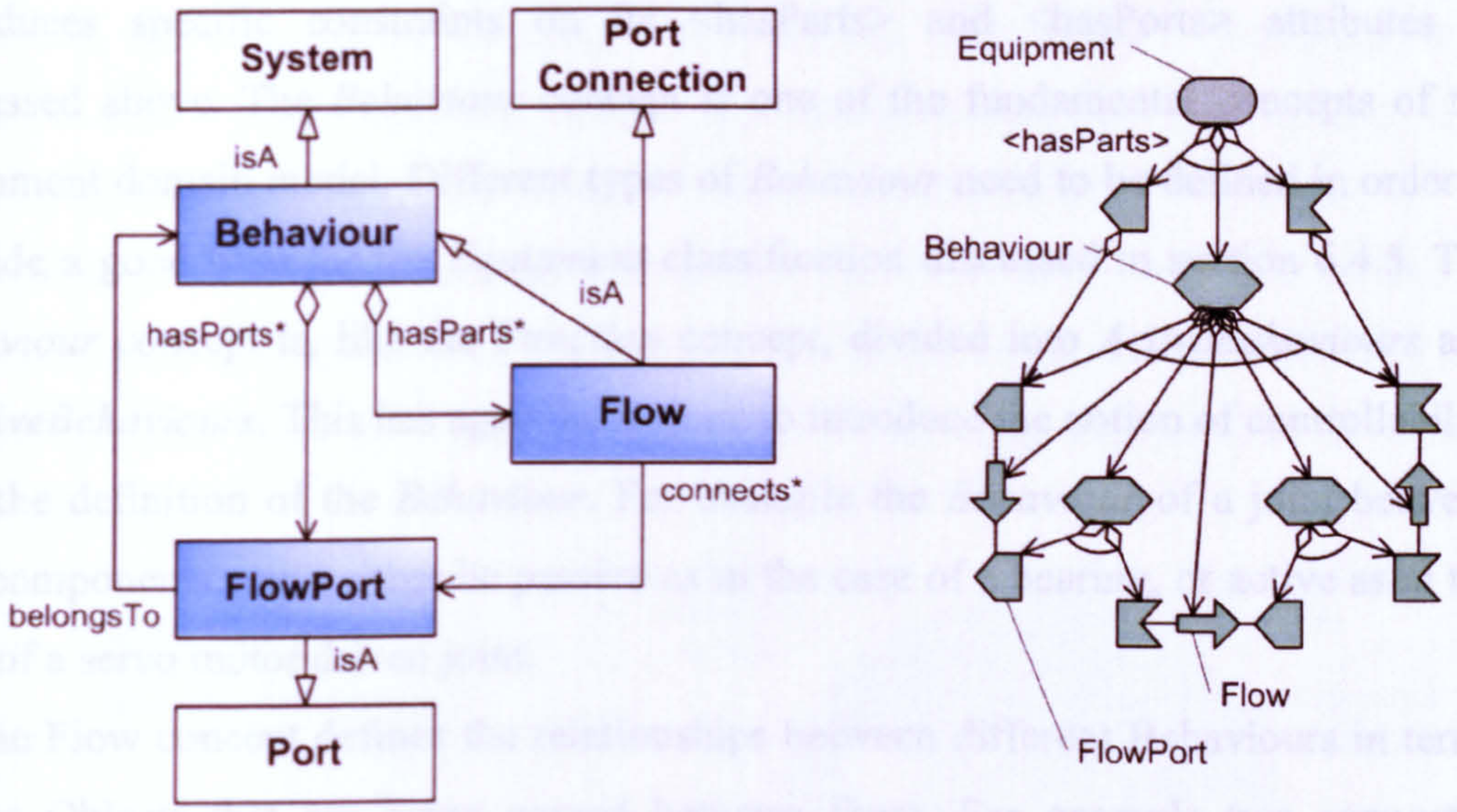


Figure 6.13 Behaviour Definition Overview

Figure 6.13 shows the principle structure of the behaviour definition model. The central *Behaviour* concept is defined as extension of the general *System* concept and defines its interactions with other *Behaviour* concepts through the *FlowPort* concept. The *Flow* concept is used to define the way in which different objects pass between different *Behaviours*. The *Behaviour* can be defined on different levels of detail using its <hasParts> attribute.

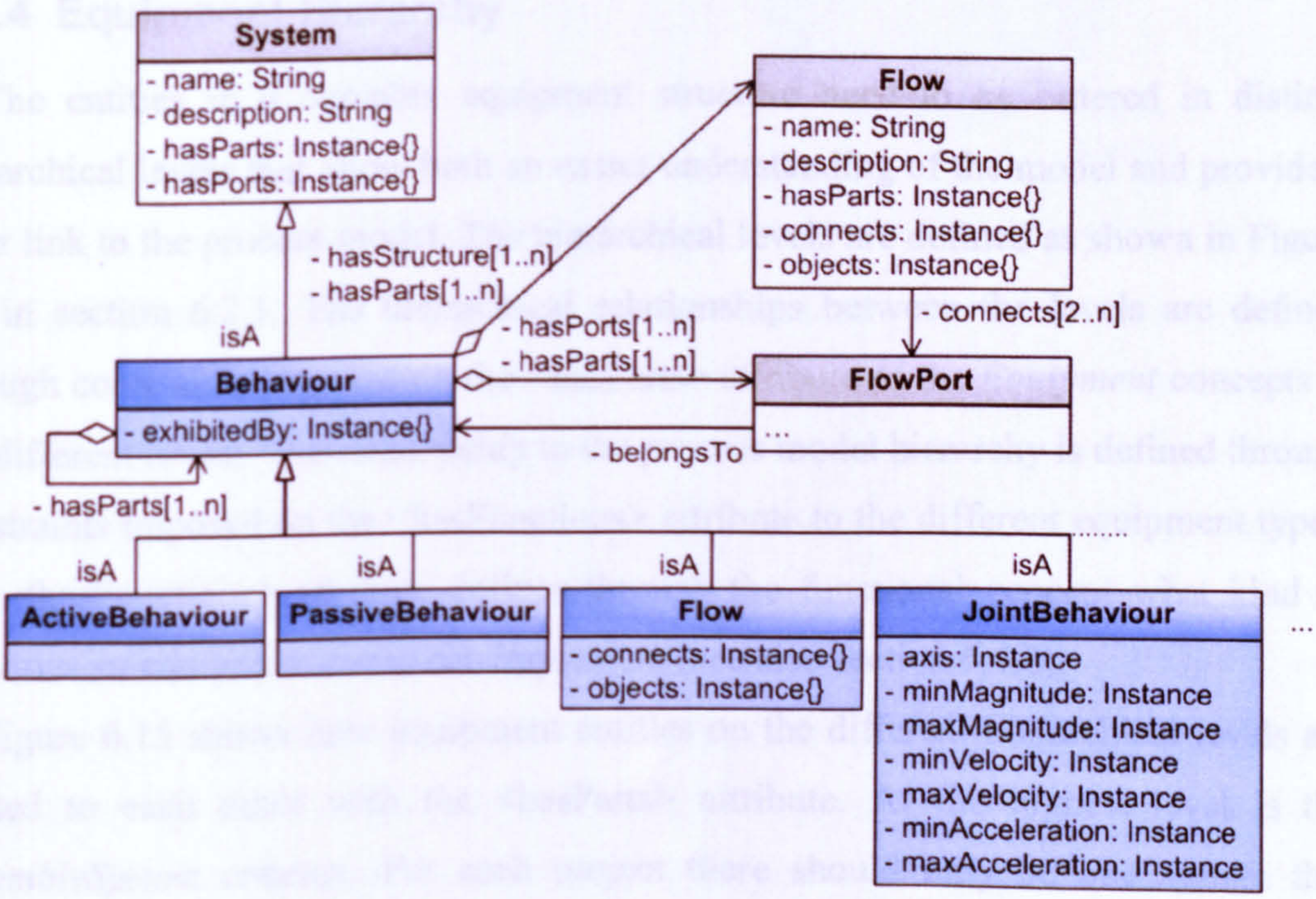


Figure 6.14 Behaviour Concept Definition

Figure 6.14 shows a more detailed definition of the *Behaviour* concept. On the highest level, the *Behaviour* concept does not have any additional attributes and only introduces specific constraints on its <hasParts> and <hasPorts> attributes as discussed above. The *Behaviour* concept is one of the fundamental concepts of the equipment domain model. Different types of *Behaviour* need to be defined in order to provide a good base for the *Equipment* classification discussed in section 6.4.5. The *Behaviour* concept is, like the *Function* concept, divided into *ActiveBehaviours* and *PassiveBehaviours*. This has again been done to introduce the notion of controllability into the definition of the *Behaviour*. For example the *Behaviour* of a joint between two components could either be passive as in the case of a bearing, or active as in the case of a servo motor driven joint.

The Flow concept defines the relationships between different Behaviours in terms of the Objects that are being passed between them. For example two connected conveyors pass components on pallets (*MaterialObjects*) between them.

The behaviour model needs to be linked to the required simulation models for the evaluation of *Equipment* entities or sets. Ideally this should take place by directly mapping the behavioural definition of the *Equipment* to specific simulation models. The simulation focuses in many cases only very specific aspects of the *Behaviour*, for example kinematics or dynamics simulation.

6.4.4 Equipment Hierarchy

The entities in a complex equipment structure need to be ordered in distinct hierarchical levels that allow both an easier understanding of the model and provide a clear link to the process model. The hierarchical levels are defined as shown in Figure 6.2 in section 6.2.1. The hierarchical relationships between the levels are defined through constraints imposed on the <hasParts> attribute of the *Equipment* concepts at the different levels. The relationship to the process model hierarchy is defined through constraints imposed on the <hasFunctions> attribute to the different equipment types. The <hasFunctions> attribute defines through the functional concept what kind of activities an equipment entity can implement (see also section 6.4.2).

Figure 6.15 shows how equipment entities on the different hierarchical levels are related to each other with the <hasParts> attribute. At the highest level is the *AssemblySystem* concept. For each project there should only be one system that contains all the equipment entities required to fulfil the required assembly process.

AssemblySystems can contain *AssemblyCells* and *EquipmentUnits* on the next lower level. *AssemblyCells* are associated to the assembly of sub-assemblies in the product structure. *EquipmentUnits* are associated with the fulfilment of operations on the process side. They normally define the transportation on the system level. *AssemblyCells* in turn can contain other *AssemblyCells*, *Workstations*, and *EquipmentUnits*. Again the *EquipmentUnits* at this level normally define the transportation. *Workstations* are associated with the achievement of elementary tasks at the process side. The putting together of individual components is done in *Workstations*.

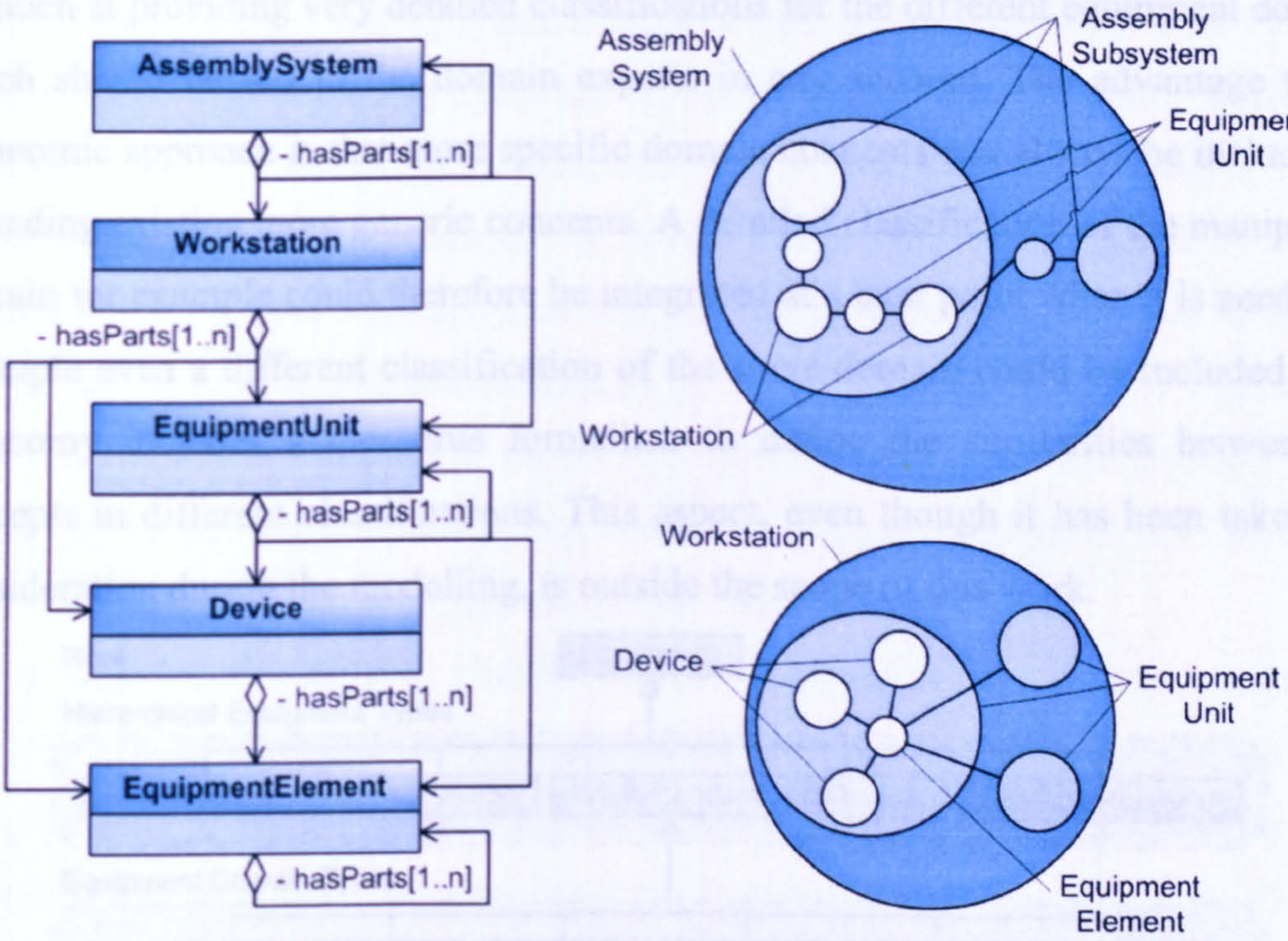


Figure 6.15 Assembly Equipment Hierarchy

The logistics or transportation takes a less dominant role on the sub-workstation levels. *Workstations* are composed of *EquipmentUnits*, *Devices*, and *EquipmentElements*. *EquipmentUnits* in this context are not restricted to transportation but on the contrary are more focused on the assembly. Dedicated *Workstations* are normally made up from collections of *Devices* that are connected with customised *EquipmentElements*. *EquipmentUnits* can be composed from *Devices* and *EquipmentElements*. They are associated with operations on the process side. *Devices* are made from *EquipmentElements*. They are normally dedicated and are associated to the fulfilment of actions on the process side. *EquipmentElements* are the corresponding concept to *Components* on the product domain side. They define the

fundamental geometric structure of all equipment entities. The definition here is restricted to their geometry and kinematic relationships.

6.4.5 Equipment Taxonomy

The different types of *Equipment* are classified based on their individual functional capabilities and based on their level in the equipment hierarchy. A hierarchical taxonomy has been defined that provides the structure for the required equipment classification. The equipment taxonomy suggested in this section is still a very high level taxonomy that aims more at structuring the domain in the general sense and not so much at providing very detailed classifications for the different equipment domains which should be left to the domain experts in any account. The advantage with a taxonomic approach is that more specific domain concepts can always be included by extending existing more generic concepts. A detailed classification of the manipulator domain for example could therefore be integrated at a later point when it is needed. In principle even a different classification of the same domain could be included if the taxonomy includes a thesaurus formalism to define the similarities between the concepts in different classifications. This aspect, even though it has been taken into consideration during the modelling, is outside the scope of this work.

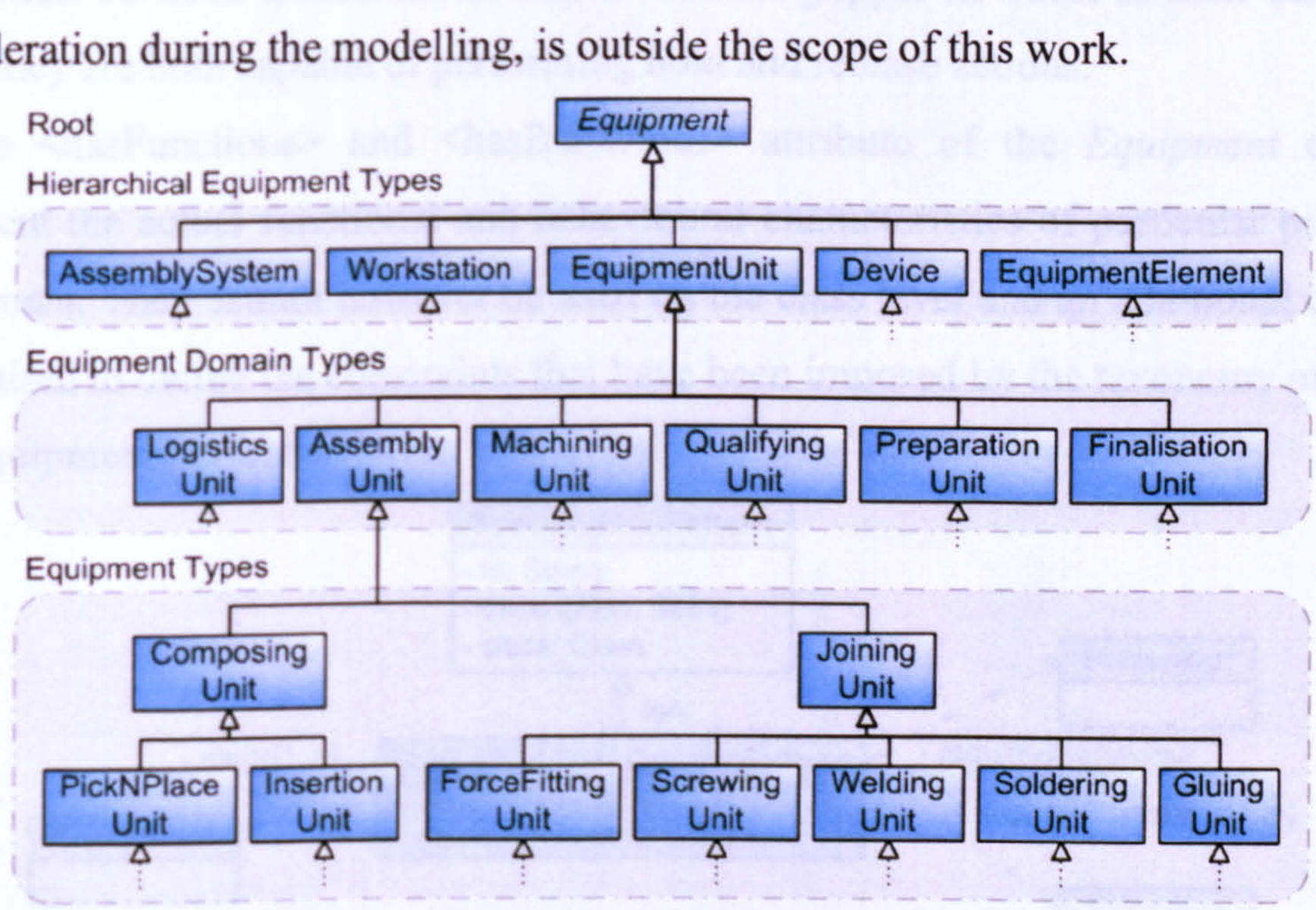


Figure 6.16 Principle Structure of the Equipment Taxonomy

The proposed equipment taxonomy classifies the equipment on the highest level of abstraction into concepts based on the distinct levels of hierarchy discussed in section 6.2.1. These concepts are: *AssemblySystem*, *Workstation*, *EquipmentUnit*, *Device*, and *EquipmentElement* (see Figure 6.15 and Figure 6.16). On the next lower level of

abstraction the taxonomy splits the hierarchical types into specific equipment domains that are associated to the activity domain types defined in chapter 5. The domain specific equipment types are specified below the equipment domain type classification.

The cornerstones of every taxonomy are the criteria that have been used to define the distinctions between the different classes at the same level of abstraction. It is furthermore required to provide a formal definition of the classification criteria for each type of equipment to achieve an unambiguous classification scheme. This formal specification of the equipment types can then be used both to understand what type a piece of equipment is and to select a suitable type of equipment for a given set of requirements. The proposed assembly equipment taxonomy is based in the first instance on their functional capabilities and in the second instance on their implementation principle which is expressed through their behaviour. For example, the distinction between a manipulator type equipment and a gripper would be that the former has to have the functional capability to perform move actions whereas the latter has to have the functional capability to perform hold and release actions. The distinction between a mechanical and a vacuum gripper however is their behaviour since they are both capable of performing hold and release actions.

The <hasFunctions> and <hasBehaviour> attribute of the *Equipment* concept represent the actual functional and behavioural characteristics of particular pieces of equipment. They cannot however be used on the class level and an additional concept is required to define the constraints that have been imposed by the taxonomy onto this two equipment attributes.

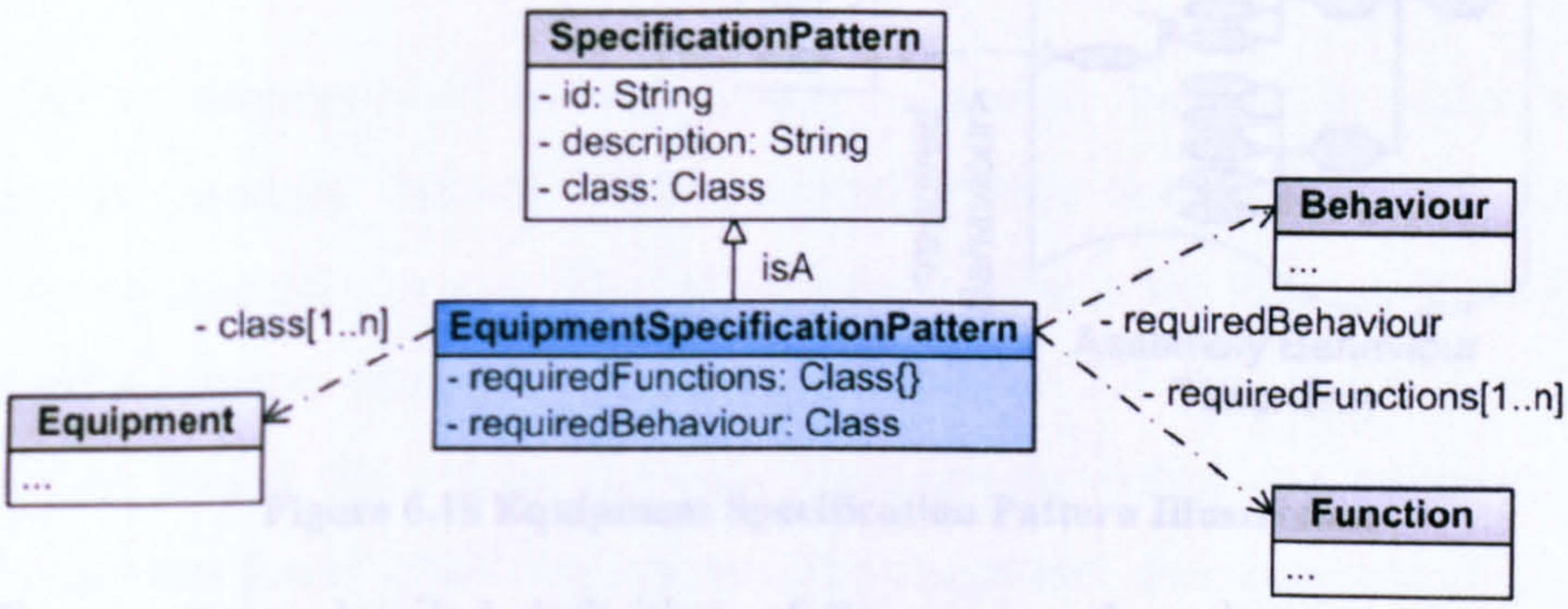


Figure 6.17 Equipment Specification Pattern Definition

The *EquipmentSpecificationPattern* concept has been introduced to provide the means for this specification of the constraints imposed on the <hasFunctions> and <hasBehaviour> attributes of the different types of *Equipment* (see Figure 6.17). For

each *Equipment* type in the equipment taxonomy there has to be an *EquipmentSpecificationPattern* instance that is linked with its <class> attribute to the *Equipment* class. The <requiresFunctions> and <requiresBehaviour> attributes are then used to define what *Function* types and *Equipment* type is required for an equipment entity to have this *Equipment* type. Figure 6.18 shows an example of *EquipmentSpecificationPatterns* that illustrates that this specification works like an AND/OR graph. Each *Equipment* type can have more than one *EquipmentSpecificationPattern* which is interpreted as an OR relationship. In each *EquipmentSpecificationPattern* can be linked to more than one *Function* which corresponds to an AND relationship. Note that the interpretation of the *EquipmentSpecificationPattern* concept is following the hierarchy of the taxonomy. The specification of lower level *Equipment* types can only be a functional and behavioural subset of its parent type.

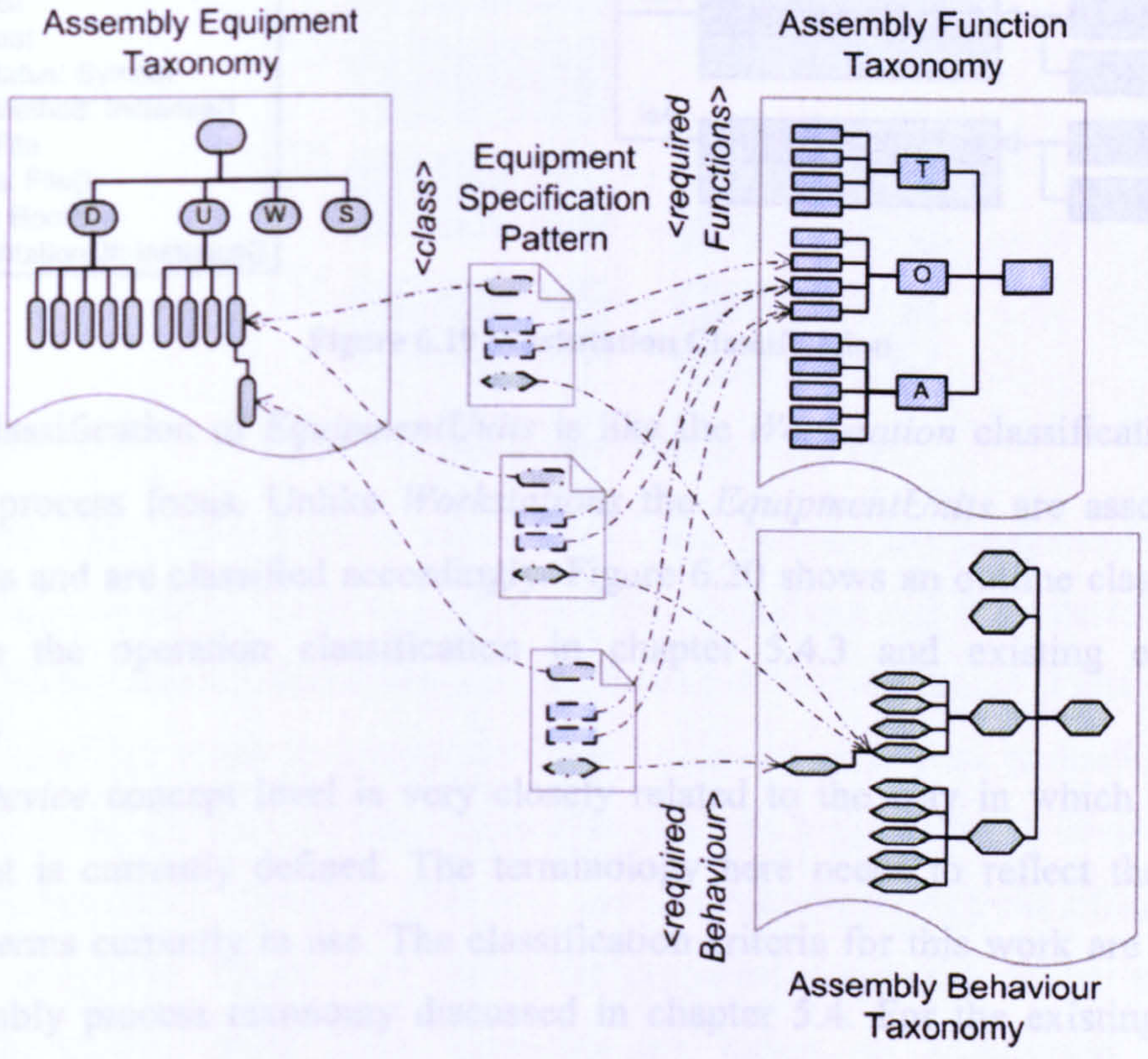


Figure 6.18 Equipment Specification Pattern Illustration

It follows a more detailed definition of the proposed equipment taxonomy. The proposed taxonomy is primarily focused on the classification of the equipment concepts at workstation and sub-workstation level in the hierarchy since the focus of this work is on the configuration of assembly workstations. The classification of

assembly systems is expected to be very abstract. For example one possible type of *AssemblySystem* could be an *AssemblyLine* or an *AssemblyCell* depending on the internal structure of the *AssemblySystem*.

The proposed classification of equipment on the workstation level is based on the different fundamental tasks workstations are designed to do. Figure 6.19 shows an overview of the proposed workstation classification. Further classification could be based on potential differences in the structure of the workstation, as for example different types of internal transport structure.

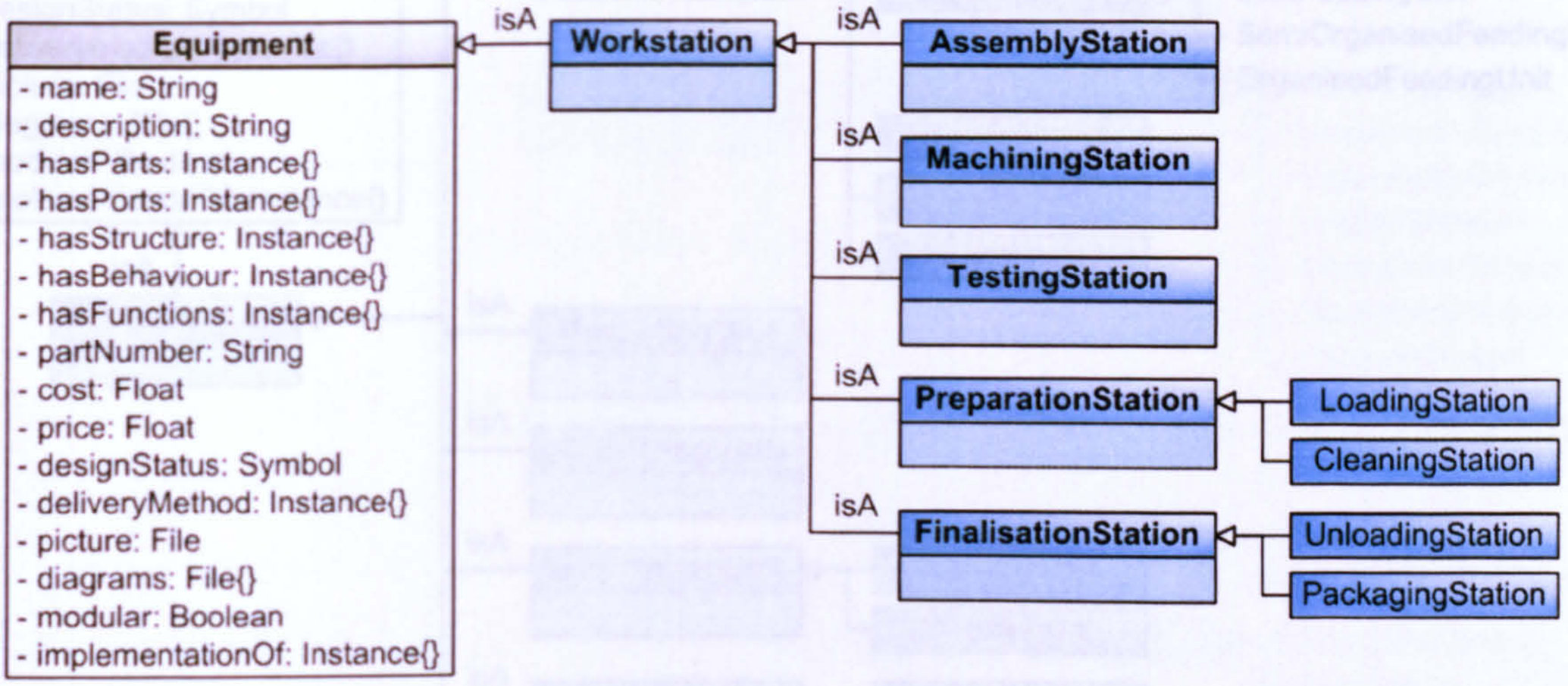


Figure 6.19 Workstation Classification

The classification of *EquipmentUnits* is like the *Workstation* classification based on their process focus. Unlike *Workstations* the *EquipmentUnits* are associated to operations and are classified accordingly. Figure 6.20 shows an outline classification based on the operation classification in chapter 5.4.3 and existing equipment solutions.

The *Device* concept level is very closely related to the way in which assembly equipment is currently defined. The terminology here needs to reflect the domain specific terms currently in use. The classification criteria for this work are based on the assembly process taxonomy discussed in chapter 5.4. For the existing domain terminologies this is not always the case although an underlying tendency towards assembly activity based classification can be recognised. For example it is common practice to group manipulators or robots together as a type of device that is used to move things within a workstation. Manipulators than are classified based on their structural characteristics and degrees of freedom (see Rampersad [99]). Both aspects are important to capture and there even is a case for arguing that a hierarchical

taxonomy is not enough but that there is a need for different classifications to be used on one interrelated model.

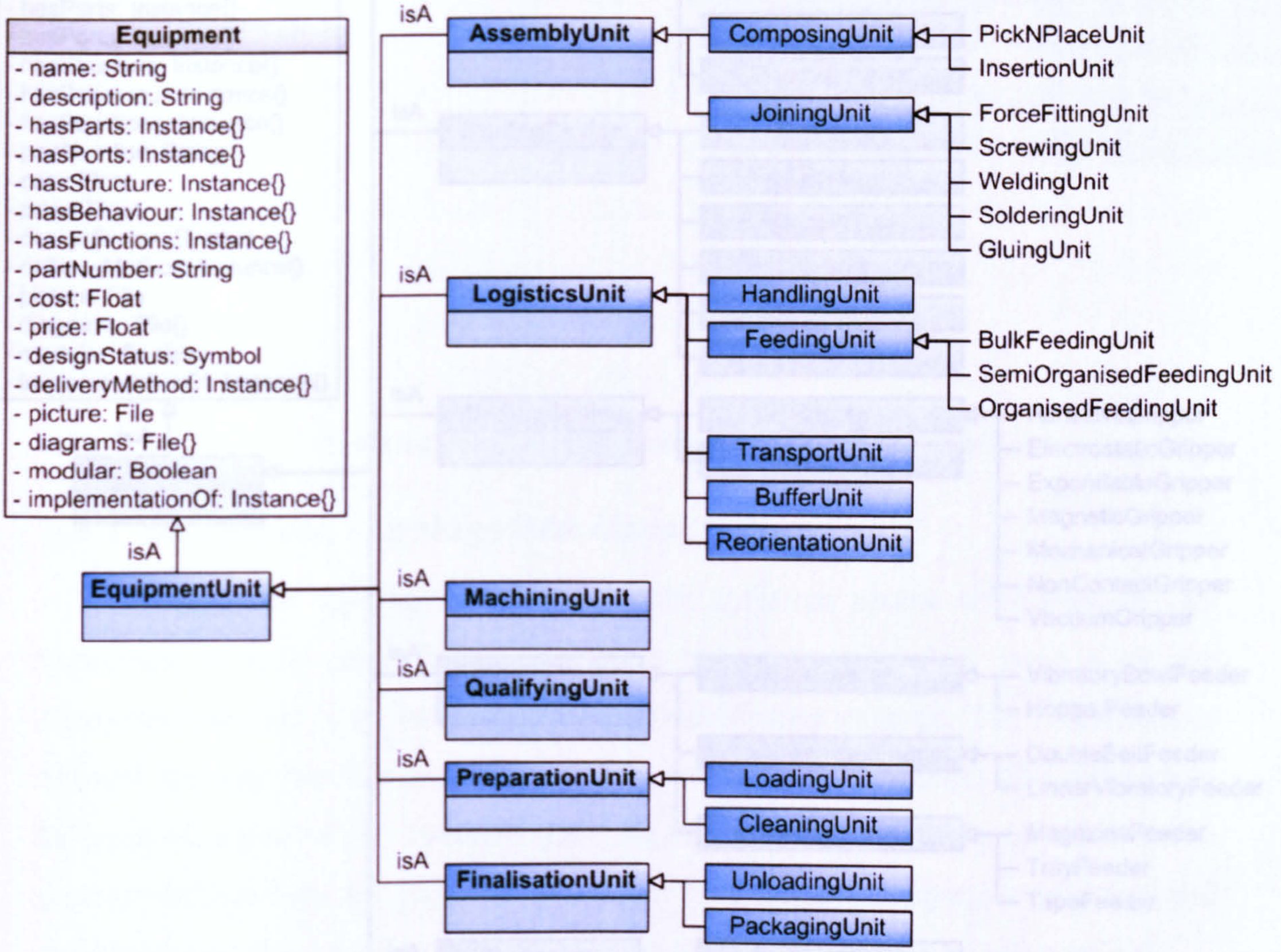


Figure 6.20 Equipment Unit Classification

Since the added complexity of such a model does not add any fundamental value to the approach suggested in this work, but is rather something that fits naturally into the proposed knowledge representation scheme, it has not been modelled at this point. For the purpose of this thesis an exemplary device taxonomy has been defined that has a high level assembly activity based classification and a lower level device domain specific classification structure. This way domain specific device classifications can easily be integrated. Figure 6.21 shows the proposed exemplary device taxonomy. The terms in the lower level taxonomy are based on Rampersad [99] and Dini, et al. [29].

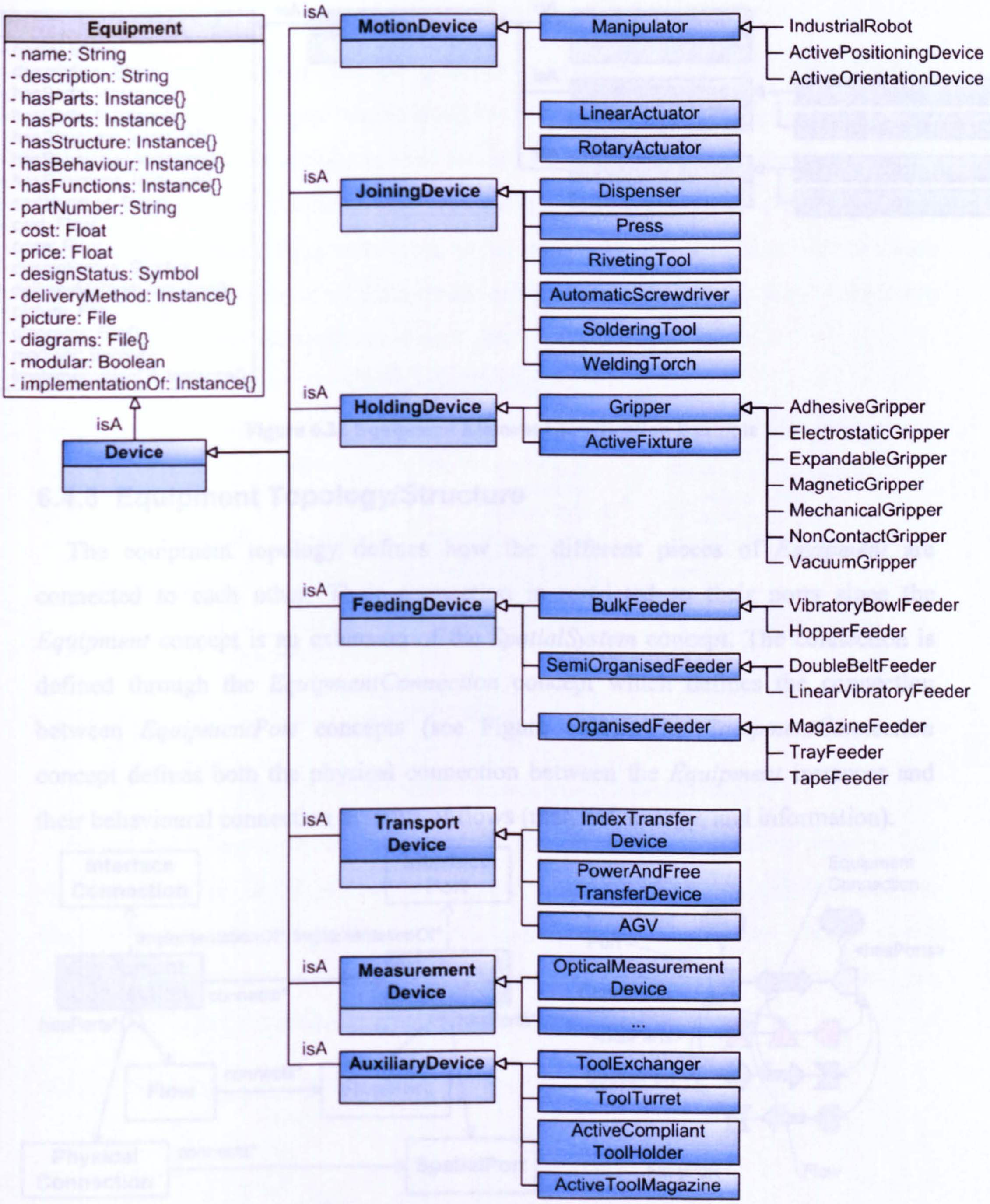


Figure 6.21 Device Classification

On the lowest level are the *EquipmentElements*. They are all passive entities that are from a concept point of view very similar to the *Component* concept on the product domain side. Figure 6.22 shows some examples of elements that occur frequently in an assembly system.

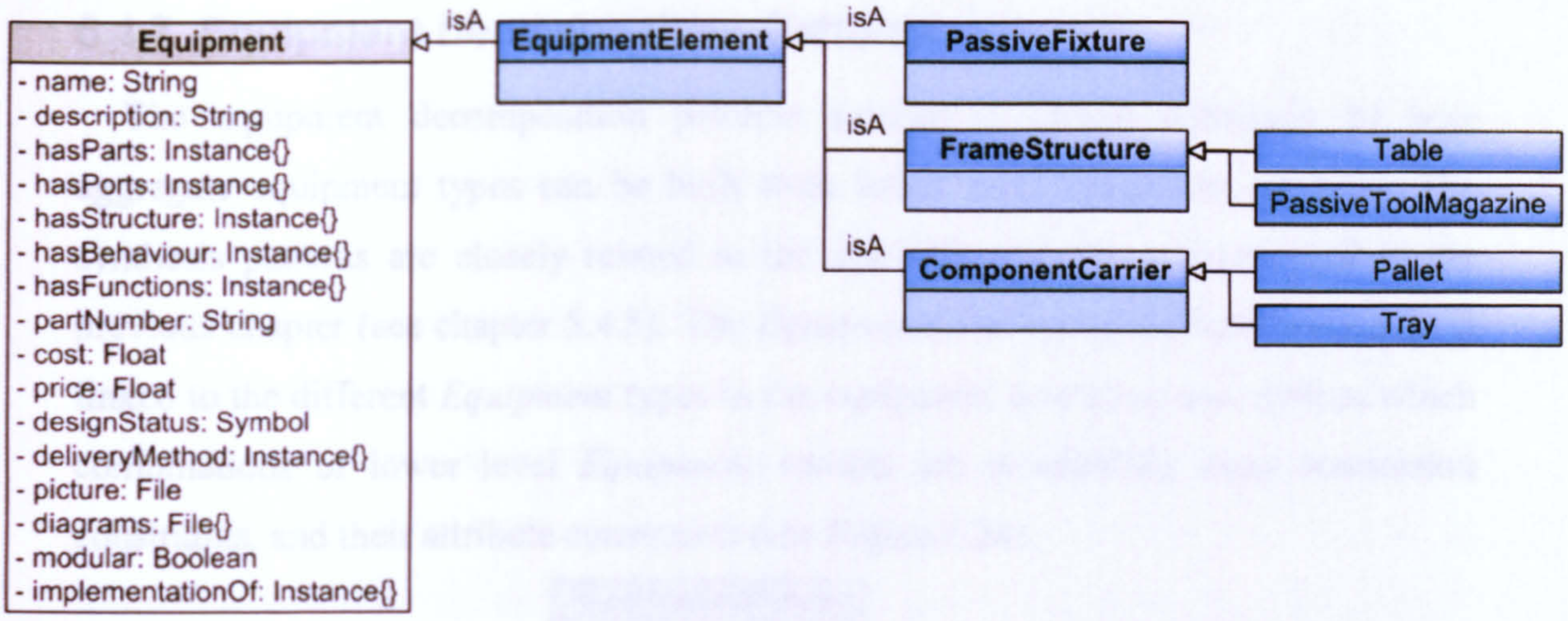


Figure 6.22 Equipment Element Classification Example

6.4.6 Equipment Topology/Structure

The equipment topology defines how the different pieces of *Equipment* are connected to each other. Their connection is restricted to their ports since the *Equipment* concept is an extension of the *SpatialSystem* concept. The connection is defined through the *EquipmentConnection* concept which defines the connection between *EquipmentPort* concepts (see Figure 6.23). The *EquipmentConnection* concept defines both the physical connection between the *Equipment* instances and their behavioural connection in terms of flows (material, energy, and information).

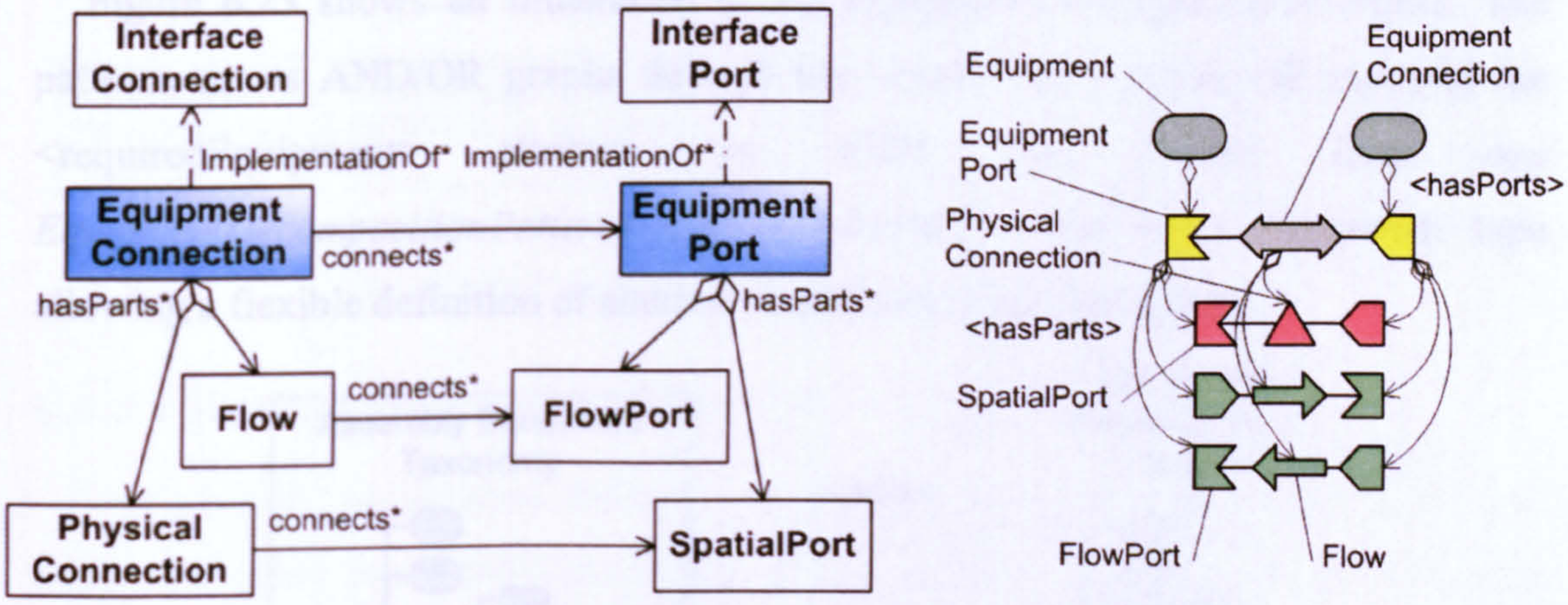


Figure 6.23 Equipment Topology Definition Overview

The *EquipmentPort* concept can be implementing an *InterfacePort* standard definition. This is required for the definition of modular equipment solutions. The *InterfacePort* concept defines the required type of *SpatialPort* and *FlowPort* (see section 6.4.8.2 for more details).

6.4.7 Equipment Decomposition Patterns

The equipment decomposition patterns provide a formal definition of how aggregate equipment types can be built from lower level equipment modules. The synthesis patterns are closely related to the assembly process patterns used in the previous chapter (see chapter 5.4.5). The *EquipmentDecompositionPattern* concept is linked to the different *Equipment* types in the equipment taxonomy and defines which combinations of lower level *Equipment* entities are permissible, their connection constraints, and their attribute constraints (see Figure 6.24).

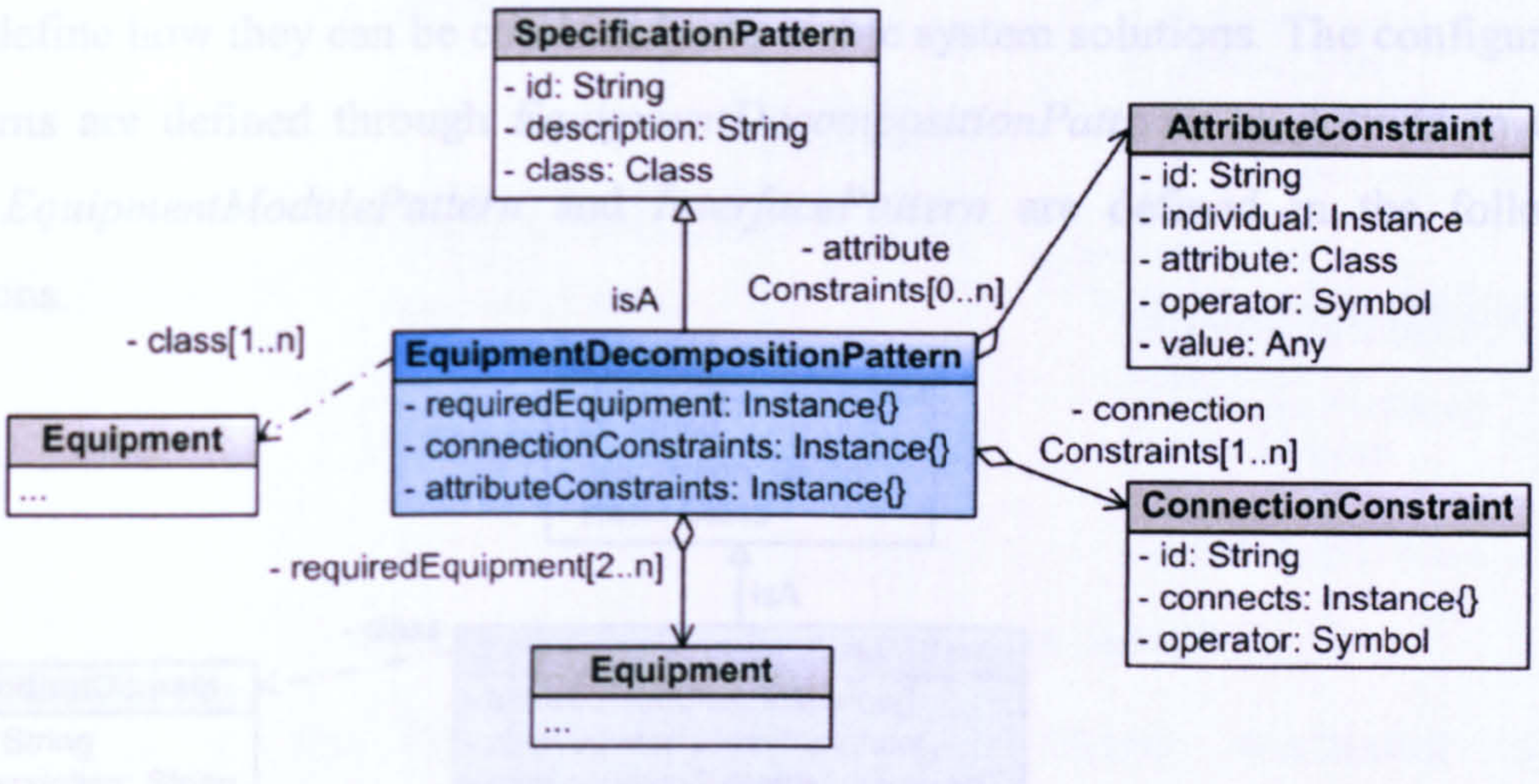


Figure 6.24 Equipment Decomposition Pattern Definition

Figure 6.25 shows an illustration of the *EquipmentDecompositionPatterns*. The patterns act as AND/OR graphs through the <class> attribute as OR part and the <requiredEquipment> attribute as AND part. More than one *EquipmentDecompositionPatterns* can be attached to the same *Equipment* type allowing a flexible definition of alternative equipment configurations.

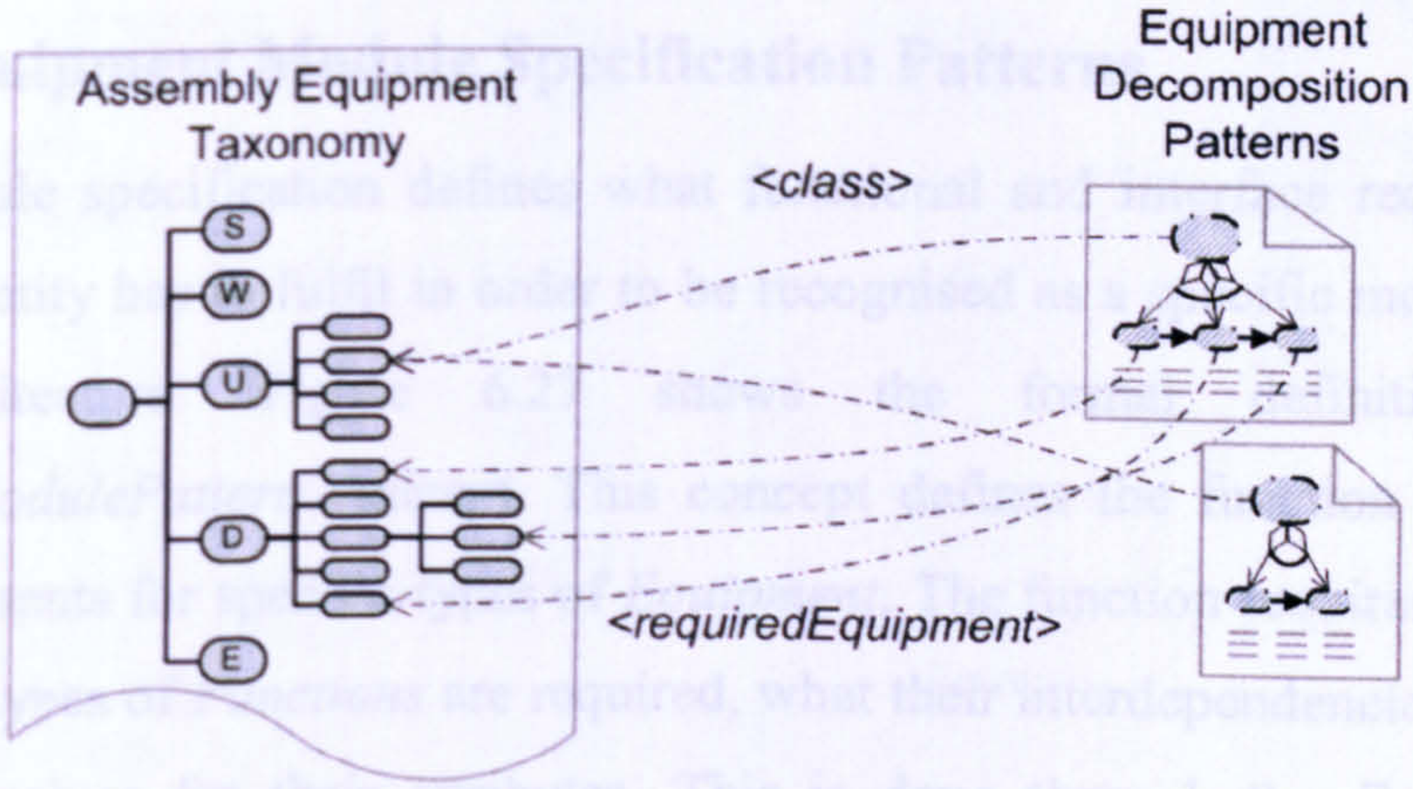


Figure 6.25 Illustration of Equipment Decomposition Patterns

6.4.8 System Architecture Definition (Configuration Patterns)

The purpose of a system architecture definition is to provide a clear structure for the specification of different assembly system instances within a given domain. The idea behind such an architecture can be compared with the combinatorial plans defined as a requirement by Pahl and Beitz [3] to cope with high degrees of complexity within problem domains.

The *EquipmentArchitecturePattern* in Figure 6.26 essentially defines the allowed set of equipment modules, their allowed interfaces, and a set of configuration patterns that define how they can be combined into viable system solutions. The configuration patterns are defined through *EquipmentDecompositionPatterns* as defined in 6.4.7. The *EquipmentModulePattern* and *InterfacePattern* are defined in the following sections.

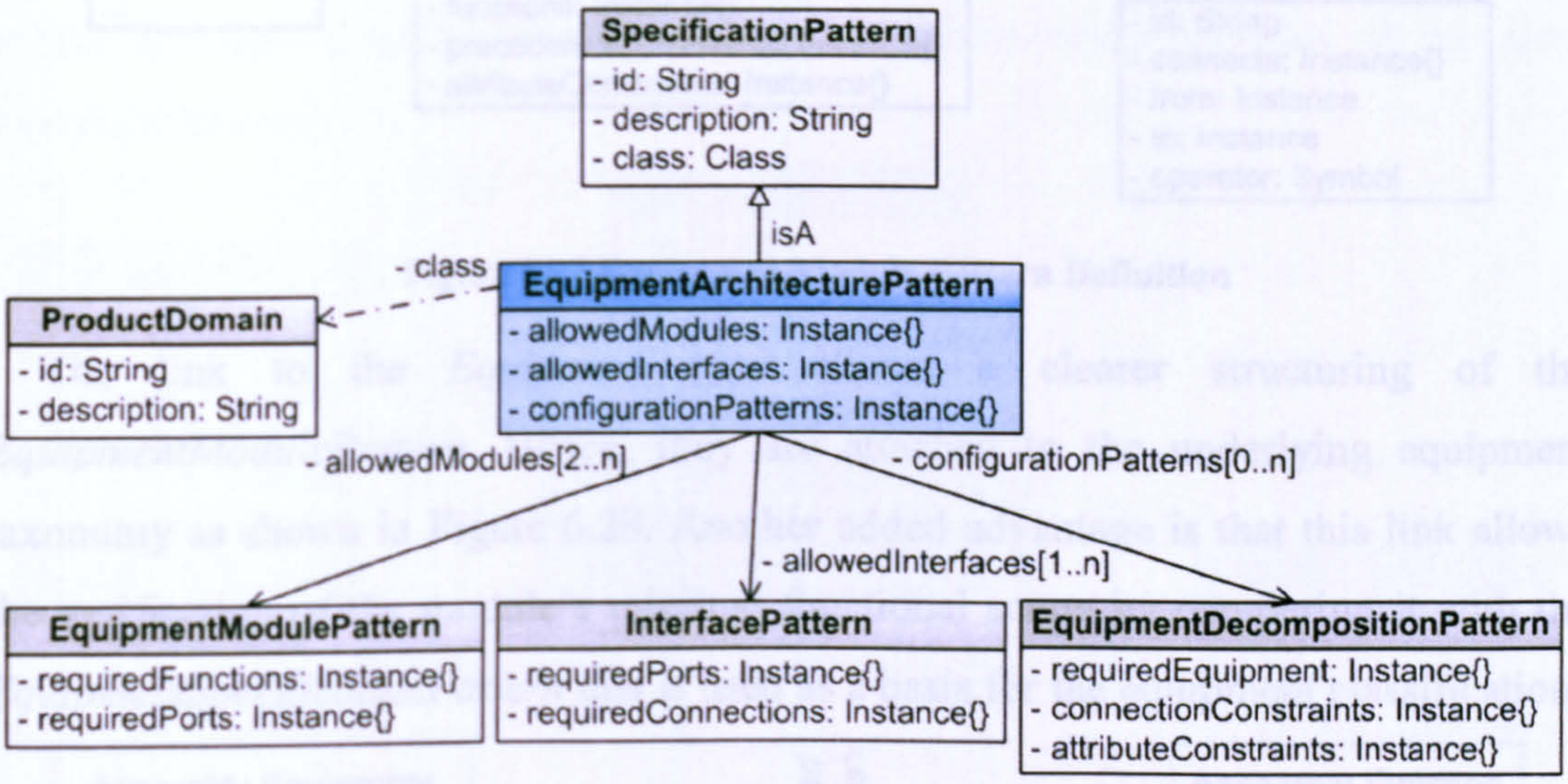


Figure 6.26 Equipment Architecture Specification Definition

6.4.8.1 Equipment Module Specification Patterns

The module specification defines what functional and interface requirements an equipment entity has to fulfil in order to be recognised as a specific module within a given architecture. Figure 6.27 shows the formal definition of the *EquipmentModulePattern* concept. This concept defines the function and interface port requirements for specific types of *Equipment*. The function requirements need to define what types of *Functions* are required, what their interdependencies are, and the permissible values for their attributes. This is done through the *FunctionPattern* concept which links *Function* classes, *PrecedenceConstraints*, and

AttributeConstraints. The interface port requirements defined which specific *InterfacePort* instances an *Equipment* instance needs to have to be defined as a particular module.

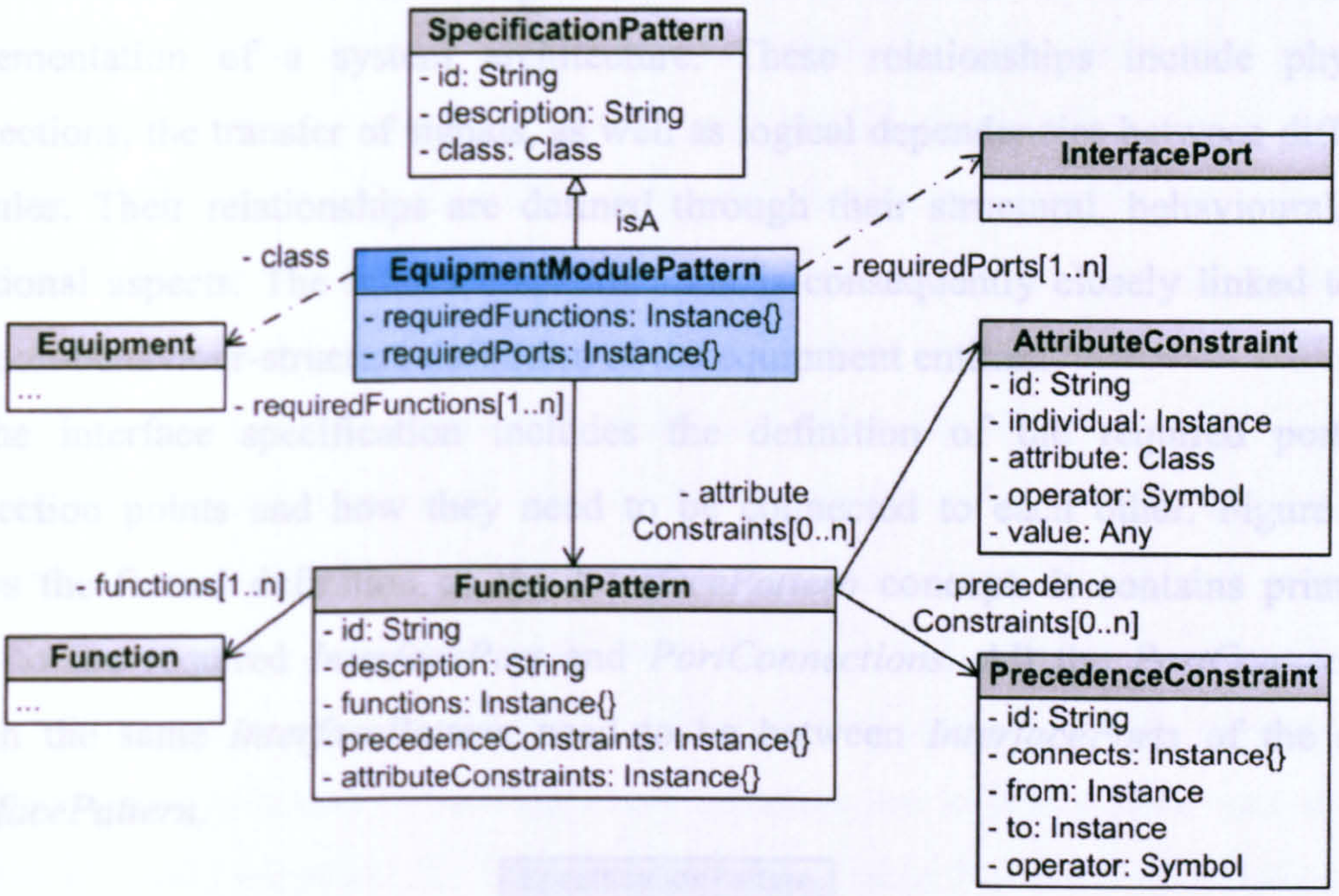


Figure 6.27 Equipment Module Pattern Definition

The link to the *Equipment* type allows a clearer structuring of the *EquipmentModulePattern*. Hence, they are attached to the underlying equipment taxonomy as shown in Figure 6.28. Another added advantage is that this link allows the verification of the module's minimal functional scope by comparing it with the *EquipmentSpecificationPattern* that is used as a basis for the equipment classification.

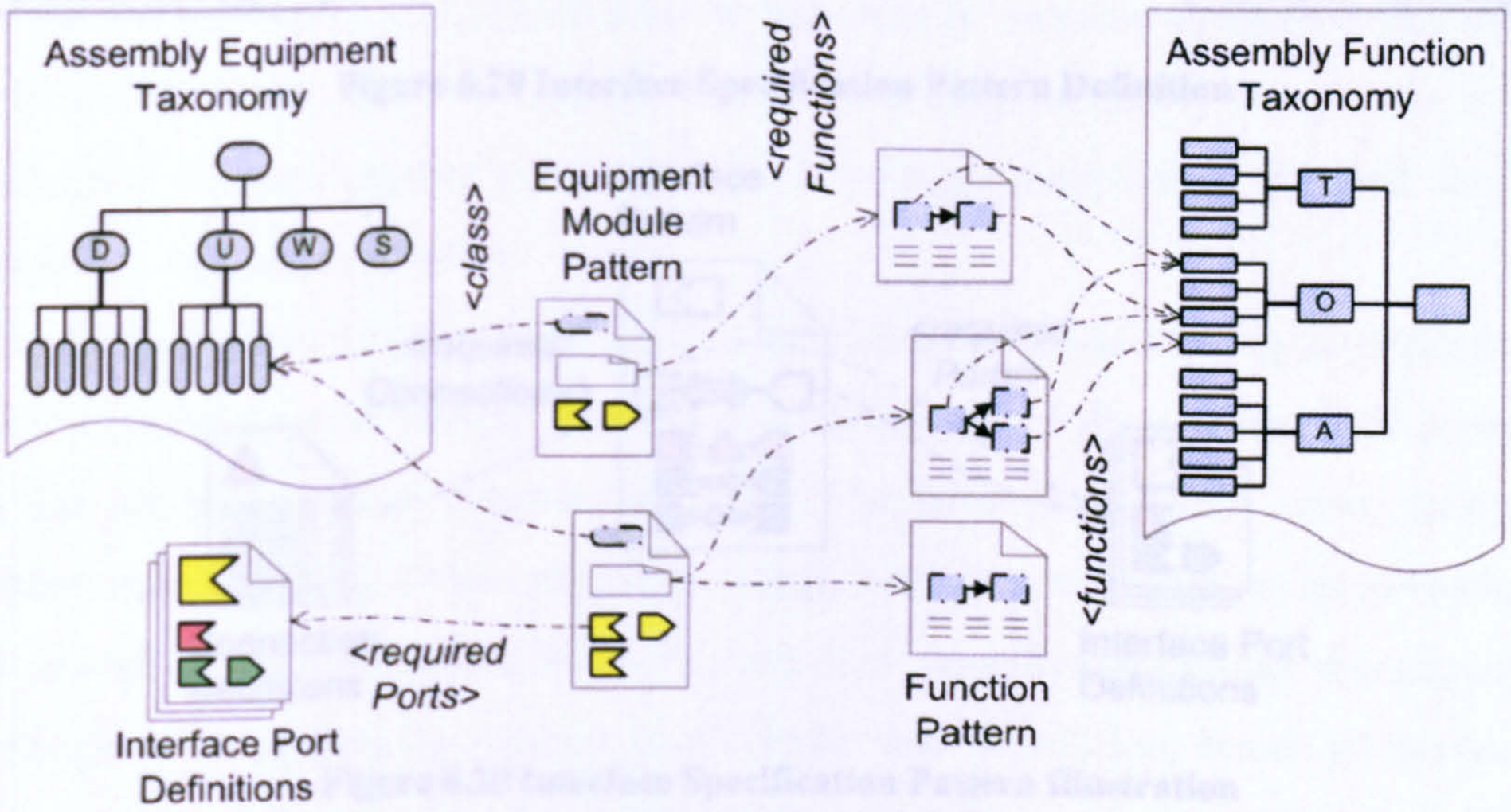


Figure 6.28 Equipment Module Specification Example

6.4.8.2 Interface Specification Patterns

The purpose of the interface specifications is to define formally the constraints placed on the different relationships that could exist between the modules within an implementation of a system architecture. These relationships include physical connections, the transfer of signals, as well as logical dependencies between different modules. Their relationships are defined through their structural, behavioural, and functional aspects. The interface specification is consequently closely linked to the function-behaviour-structure definition of the equipment entities.

The interface specification includes the definition of the required ports or connection points and how they need to be connected to each other. Figure 6.29 shows the formal definition of the *InterfacePattern* concept. It contains primarily links to the required *InterfacePort* and *PortConnections*. All the *PortConnections* within the same *InterfacePattern* need to be between *InterfacePorts* of the same *InterfacePattern*.

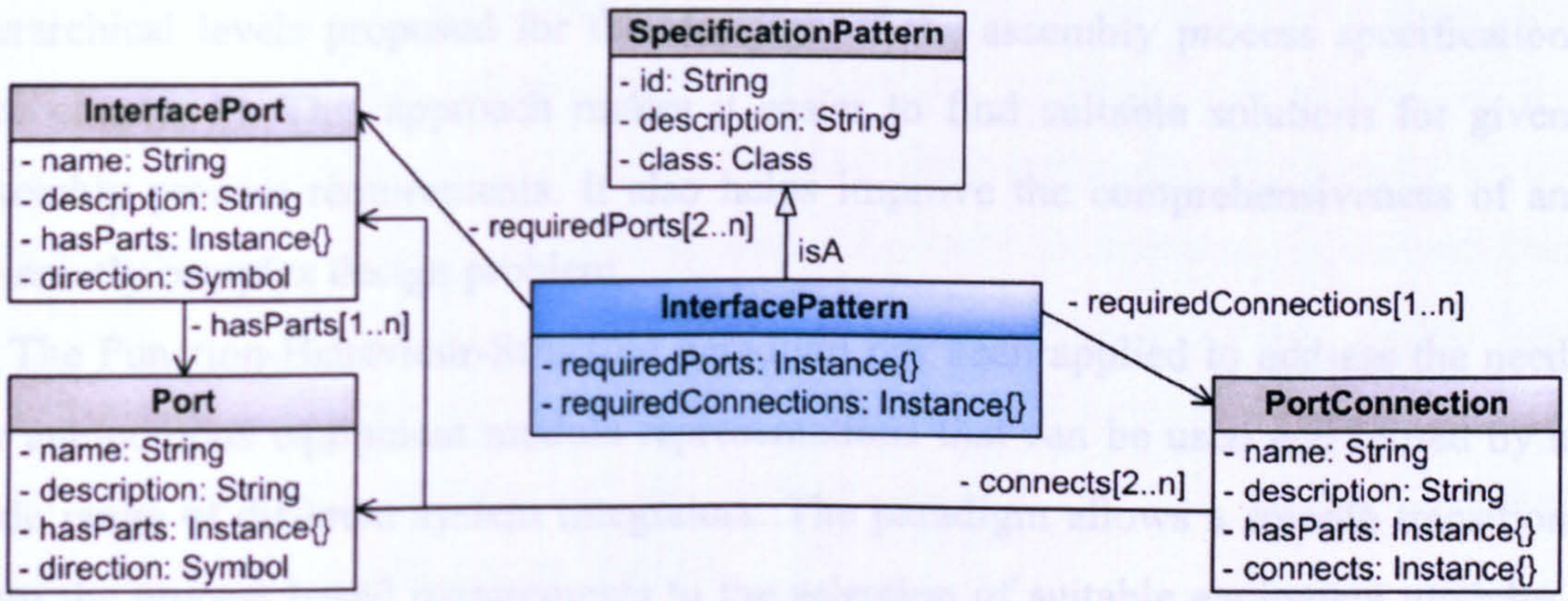


Figure 6.29 Interface Specification Pattern Definition

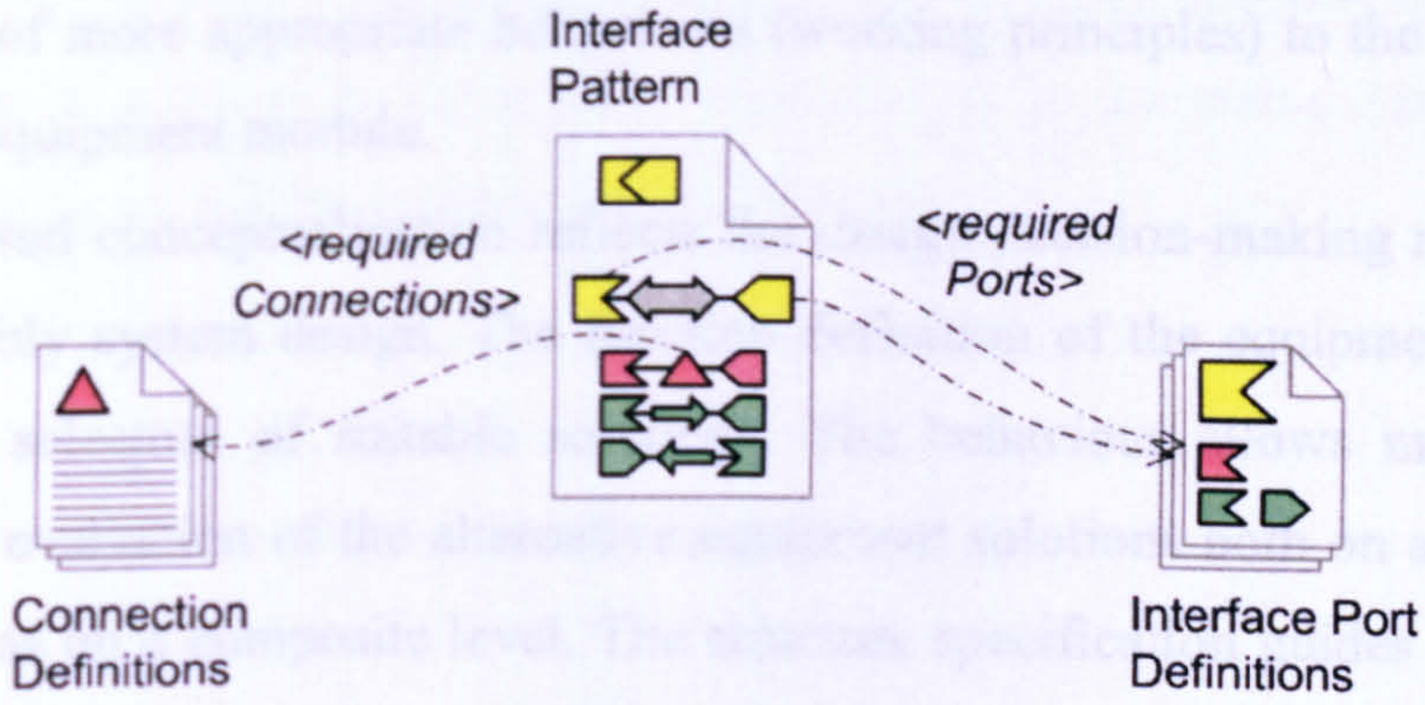


Figure 6.30 Interface Specification Pattern Illustration

6.5 Summary

In this chapter the assembly equipment domain conceptualisation for the design of modular assembly systems proposed as part of ONTOMAS has been defined and discussed. The central concept of the domain ontology is the *Equipment* concept which is defined on a temporal, logical, and spatial plane using the Function-Behaviour-Structure paradigm. The proposed formalisms enable an effective equipment selection and integration during the design of modular assembly systems. One of the key enabling factors is a clear heuristic interpretation of different *Equipment* classes and their functional capabilities. An *Equipment* taxonomy has been proposed that features Equipment classes for the definition of a clear hierarchical equipment structure. Furthermore, the taxonomy has been linked to dynamic patterns that define the difference between the different *Equipment* classes.

A new hierarchical structure for the specification of modular assembly systems has been proposed. The levels of the *Equipment* structure are logically associated to the hierarchical levels proposed for the structure of the assembly process specification (see chapter 5). This approach makes it easier to find suitable solutions for given assembly process requirements. It also helps improve the comprehensiveness of an inherently complex design problem.

The Function-Behaviour-Structure paradigm has been applied to address the need for autonomous equipment module representations that can be used and reused by a wide range of different system integrators. The paradigm allows a smooth transition from the process based requirements to the selection of suitable equipment modules. The transition occurs from the process based functional capability description, over the selection of more appropriate behaviours (working principles) to the final choice of a specific equipment module.

The proposed conceptualisation reflects the design decision-making requirements during assembly system design. The function definition of the equipment is mainly used for the selection of suitable solutions. The behaviour allows more detailed selection and evaluation of the alternative equipment solutions both on an individual level as well as on a composite level. The structure specification guides the physical integration of the equipment modules into a wider system solution. Hence all the three major aspects of the design decision-making process are covered. The Function-Behaviour-Structure paradigm is envisaged to also reduce the definition effort of

equipment providers since they could utilise predefined function and behaviour "building blocks".

The Function-Behaviour-Structure paradigm has also been extended to include the specific constraints arising from the modular system domain. Formalisms for the specification of equipment module standards have been proposed and applied. The specification prescribes the functional and interfacing capabilities an equipment module type has to fulfil to participate in a given system platform.

Domain specific *Patterns* have been proposed as part of the ONTOMAS framework to support the decision-making process during the assembly equipment selection and integration. Patterns for the classification of equipment types, for the mapping of assembly process-based requirements to functional capabilities, and for guiding the equipment decomposition and integration process, have been proposed. This allows a very dynamic definition and adaptation of the design knowledge available within the domain.

The specification of the equipment has been based on system theory principles which reflect the basic characteristics of modular systems. All aspects of the equipment conceptualisation have been based on these principles which make it very easy to adapt and maintain resulting models. The same fundamental formalisms that have been used to define the product and assembly process domain models (see chapters 4 and 5 respectively) have been used to define the equipment domain model. This improves the re-usability of the decision making methods across the whole modular assembly system design domain.

The definition of the assembly equipment domain model concludes the proposed modular assembly system conceptualisation of the ONTOMAS framework. In the next chapter the application of the three domain models in an integrated design-decision making environment will be described and discussed.

7 Integrated Assembly Process and Equipment Specification Method

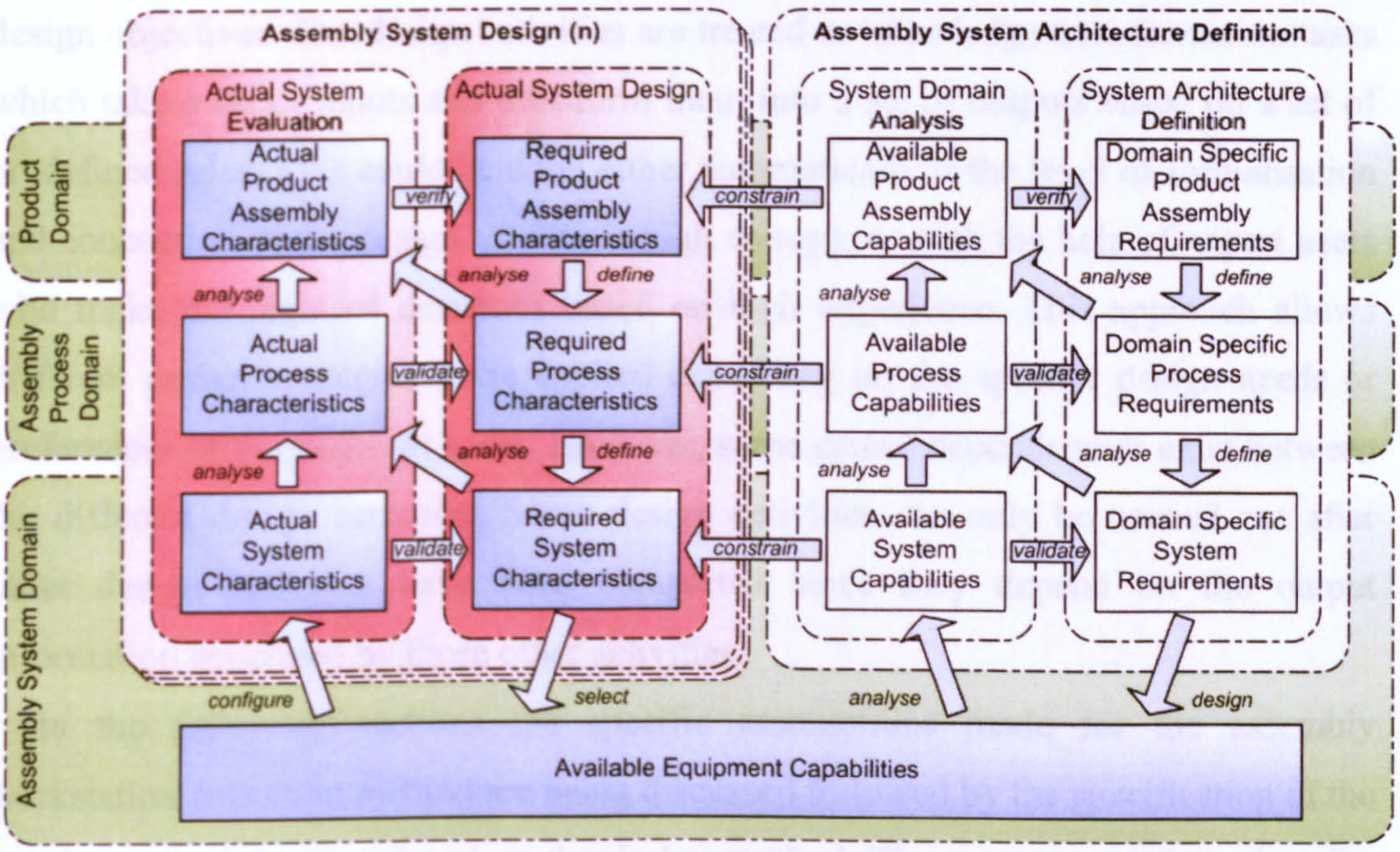


Figure 7.1 Integrated design method overview

7.1 Introduction

This chapter looks at the application of the proposed ontology framework for the design of modular assembly workstations. The primary objective is the integrated definition of the assembly process and the selection of its enabling set of equipment modules. The transformation of the product based user requirements into process based system requirements has been addressed by the work of Hirani [51].

Figure 7.1 shows an overview of the design activities addressed in this chapter. It also illustrates the link between the proposed ONTOMAS framework and the design activities that utilise its different domain ontologies (see chapter 4 to chapter 6). The focus of the integrated design method discussed in this chapter is only on the development and adaptation of individual assembly workstation solutions. All the

solutions are based on already existing equipment modules that are defined in the equipment library. Particular focus has been placed on the design of assembly workstations since the transition from system level design approaches to device based configuration approaches has not been fully explored. The right configuration of assembly workstations is also the most intricate activity during the design and adaptation of suitable assembly system solutions.

The design process of modular assembly workstations has been split into a set of design activities that can be used on different combinations to achieve the overall design objectives. The design activities are treated as knowledge transformation tasks which take a set of inputs and transform them into a set of outputs based on a set of predefined rules. This could be done either automatically if the level of formalisation and conception of the design activity is high enough, or with the help of expert users who make the required decisions based on their experience. This approach allows different design strategies to be applied depending on the specific design needs or preferences of the different users. However, some causal dependencies exist between the different design activities. Some design activities can only be carried out after other design activities have been completed since they depend on the output information generated by those other activities.

In the following sections the specific assumptions made for the assembly workstation definition method are being discussed followed by the specification of the requirements that were placed on the design method. The next two sections describe the integrated assembly process and assembly equipment specification method with primary focus on how the concepts of ONTOMAS can be applied to support the design decision-making process. Finally the chapter is concluded with a brief summary of the key advantages posed by the proposed design approach. Throughout the whole chapter the design activities are modelled using the notations defined by the CommonKADS framework (Schreiber, et al. [114]).

7.2 Assumptions

A set of assumptions has been made that defines the specific aspects of the assembly system design process which are being addressed in this work. Since this work is focused on the design of modular assembly workstations it was assumed that the system requirements have already been defined to this level of detail. This does not mean, however, that the workstation design process should not influence the

design of the overall system. On the contrary, this feedback is more than desired. The resulting iteration between the system level design and the workstation configuration is, however, outside the limited scope of this work.

It has also been assumed that all the domain specific requirements have already been translated into product, process, and equipment specific classifications and constraints. This work is reflected in the definitions in chapters 4 to 6 of this work. Specifically these assumptions include:

- The product is **fully defined** and not subject to change (static). The product definition includes: product structure, component definitions, and component relationships.
- Overall non process based system requirements are **fully defined** and not subject to change (static). They include: project constraints, process constraints, equipment constraints, and environment constraints.
- The assembly process requirements have already been defined on work station level. That means that assembly tasks and their precedence constraints have already been allocated to abstract workstation definitions. This is the result of traditional assembly planning.
- A modular assembly workstation architecture and a set of equipment modules exist that delivers different sets of assembly capabilities.
- Each equipment module has its own control capabilities and can be integrated to perform wider tasks.
- Simulation and performance evaluation tools exist that can provide the required feedback to the design environment.
- There exists a generic description of assembly activities defining the constraints between the different types of activities.
- Predictive evaluation methods exist that can provide the required feedback to the main design decision environment.

7.3 Requirements

This section lists the key requirements an integrated design method needs to satisfy. Particular attention has been given to the maintenance of links between the different domains.

- Responds dynamically to requirements changes

- Should enable a high degree of automation using knowledge and optimisation base approaches
- Enable tracking of design decision making processes between different domains
- Allow multiple users to interact in their preferred form with the environment
- Enable multi-vendor participation for the equipment suppliers
- Hierarchical problem decomposition and synthesis design approaches
- Early consideration of system design constraints arising from module availability
- Enable the exploration of different solution alternatives on demand
- Provide methods for the propagation of change
- Allow different design strategies to be employed to solve the problem
- Incorporate fixed validation and "milestones" for critical stages during the design process. E.g. the stage at which a quotation can be provided with a reasonable accuracy
- Be open to changes in the underlying design and business rules and strategies.
- Allow the expertise of different domain experts to be utilised to its highest potential.

7.4 Requirements Driven Assembly Process Specification

The assembly process specification is fundamentally based on a hierarchical decomposition approach guided by rules that capture the expert knowledge of the domain and the constraints of currently available equipment module implementations (top-down approach).

The decomposition is based on fixed associations between predefined hierarchical process levels (see chapter 5.4.2). These hierarchical levels have been integrated into a process taxonomy that classifies different equipment types based on their functional capabilities and behavioural characteristics (see chapter 5.4.3). The relationships between the hierarchical levels are defined through process decomposition patterns. One or more process decomposition patterns can be applied to any one type of activity. They specify the types of sub-activities, their temporal relationships, and their parametric constraints (see chapter 5.4.5 for more details).

The process specification takes place on the conceptual level and allows the dynamic decomposition of activities. The order in which higher level processes need to be decomposed is not fixed by the method. Different strategies can be applied to cater for the specific preferences or needs of different expert users. The And-Or-Graph structure of the conceptual process definition model allows also several solution alternatives to be explored and compared at the same time. This way, decision alternatives can be dynamically maintained and explored at the liberty of the decision maker.

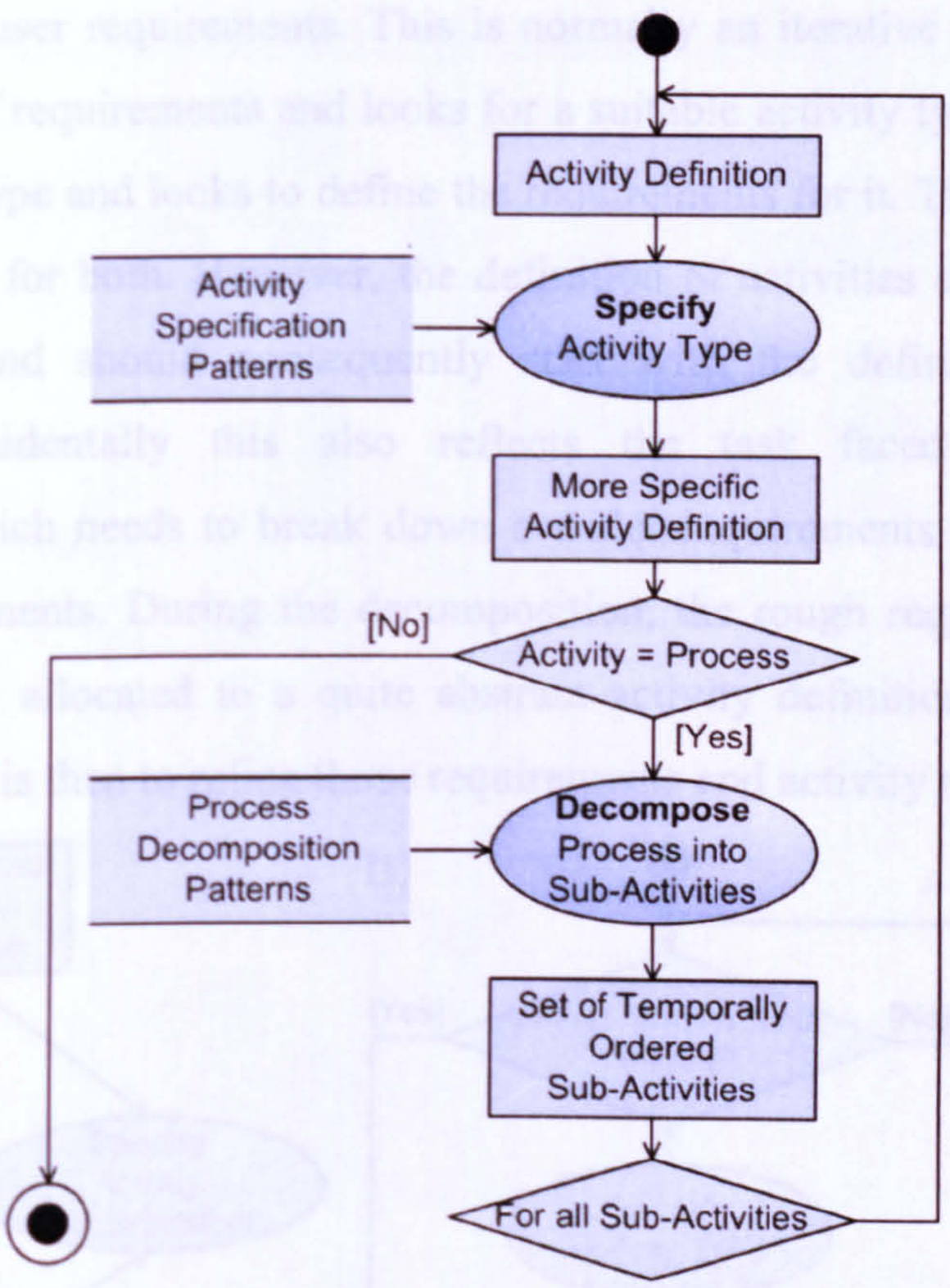


Figure 7.2 Assembly Process Specification Overview

The assembly process specification process is split into two tasks: the specialisation of the required activity and the decomposition of the activity into sub-activities. The activity specialisation task addresses the specification of the required activity type and its required parameters. The process decomposition task looks at how a complex activity can be decomposed into lower level sub-activities. Both tasks are semi independent. Their only causal relation is that the activity specialisation task should take place before the decomposition. This separation of the specification activities also reflects the And-Or-Graph definition of the conceptual process model.

Adding new decompositions (process sequences) is adding OR branches to the graph, and adding sub-activities into the sequence is adding AND branches to the graph.

Figure 7.2 shows an overview of the assembly specification process. The arrows only indicate the principle relationships and do not define absolute decision orders. The following sections discuss the two process specification tasks in more detail.

7.4.1 Activity Specialisation Task

The activity specialisation task looks at defining the required activity type of a set of product based user requirements. This is normally an iterative process that either starts with a set of requirements and looks for a suitable activity type, or starts with a required activity type and looks to define the requirements for it. The framework does in principle allow for both. However, the definition of activities should normally be purpose driven and should consequently start with the definition of a set of requirements. Incidentally this also reflects the task faced by the process decomposition which needs to break down a wider requirements space into smaller groups of requirements. During the decomposition, the rough requirements space is being defined and allocated to a quite abstract activity definition. The role of the specialisation task is then to refine those requirements and activity types.

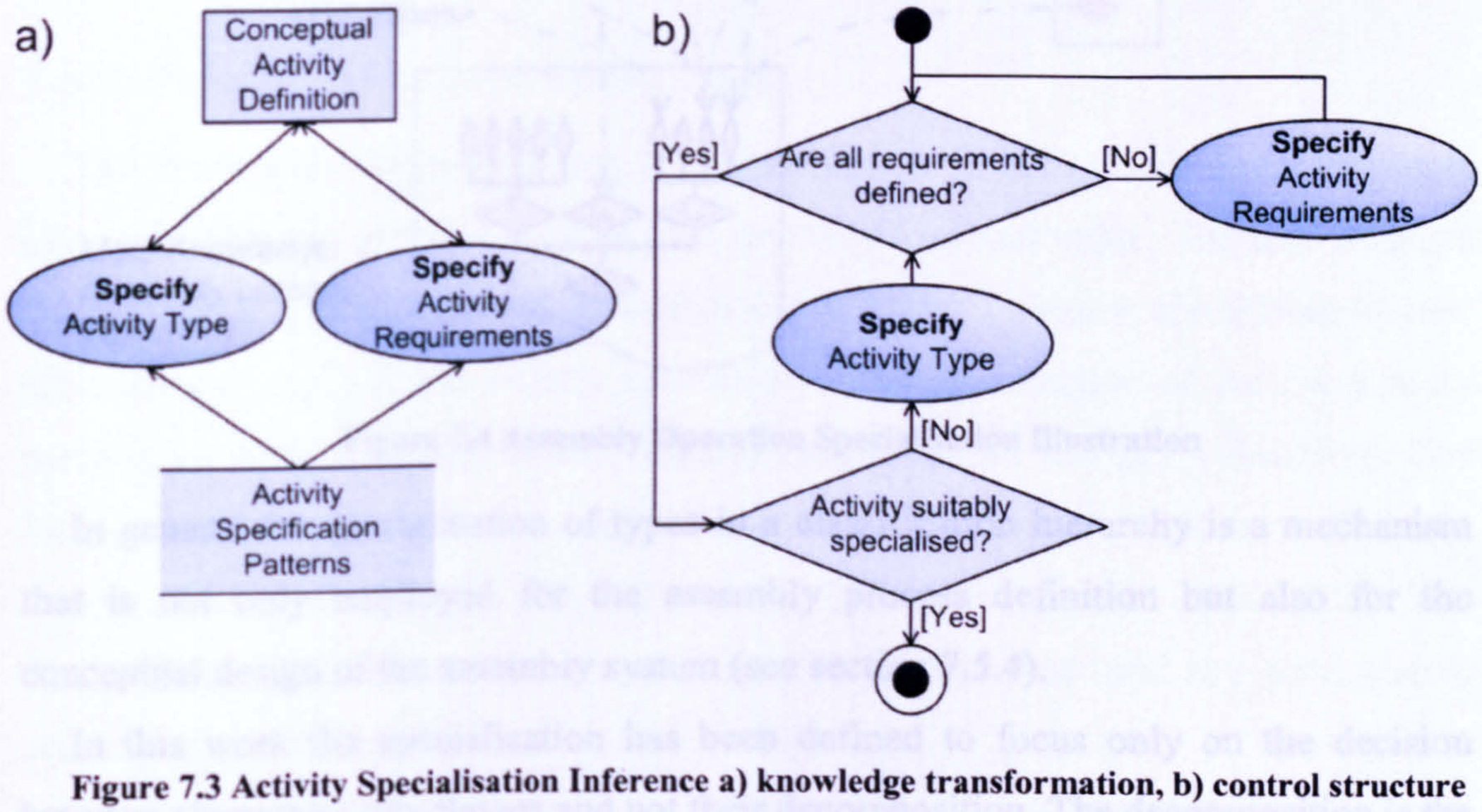


Figure 7.3 Activity Specialisation Inference a) knowledge transformation, b) control structure

Figure 7.3 shows both the knowledge transformation aspects (a) and the iterative control strategy (b) for the activity specialisation task. Both the specification of the activity type as well as the specification of the activity requirements take an existing activity definition as a dynamic input and generate a more specific version of this

definition as an output. Both specifications use the activity type specification patterns to guide the specialisation.

Figure 7.4 shows an illustration of how the specialisation is guided by the activity type specification patterns. The example of the assembly operation specification is used since it is the most central for the assembly process definition. In principle, the activity type specification pattern can be used to both drive the type selection and the definition of the requirements parameters. For this work, however, these patterns have only been used to validate the consistency between the specified activity type and the defined requirements. This aspect is further illustrated in chapter 8.

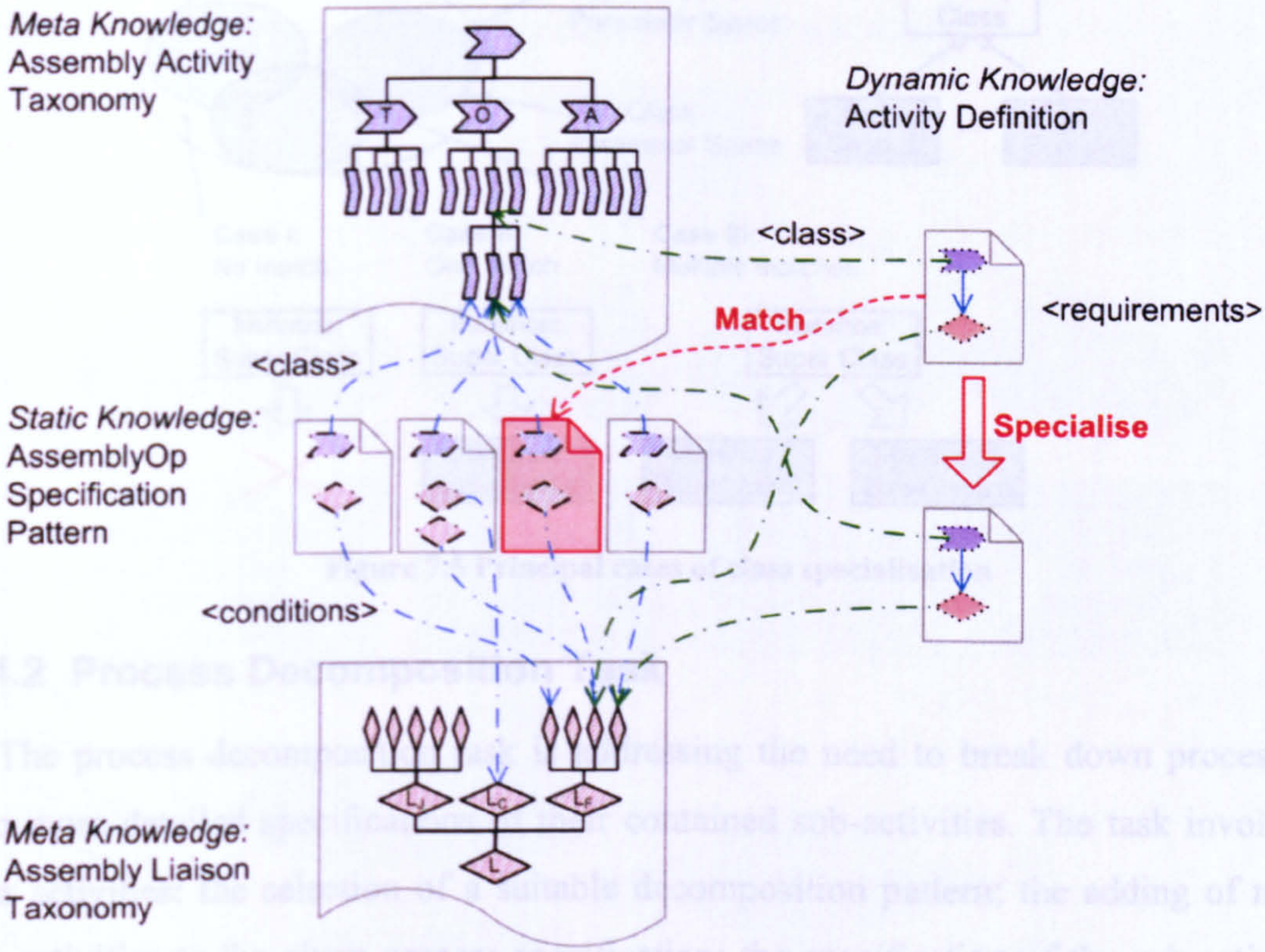


Figure 7.4 Assembly Operation Specialisation Illustration

In general the specialisation of types in a classification hierarchy is a mechanism that is not only employed for the assembly process definition but also for the conceptual design of the assembly system (see section 7.5.4).

In this work the specialisation has been defined to focus only on the decision between alternative sub-classes and not their decomposition. The decomposition is the responsibility of a separate mechanism (see section 7.4.2). The decision between alternative sub-classes has to be based on the parameter space of the super class. Specialisations of the *ActivitySpecificationPatterns* are used to define the parameter spaces of the different types of sub-concepts.

The type specialisation activity is matching the existing *ActivitySpecificationPatterns* against the parameters of the currently defined higher level instances. The higher level instances are being specified depending on the existing matches. Three general cases can occur during the matching: no match (I), one match (II), and multiple matches (III). Figure 7.5 schematically illustrates the matching process. In case I no specialisation is possible. In case II exactly one specialisation exists that can be applied immediately. In case III more than one specialisation is possible and a decision needs to be made which to use.

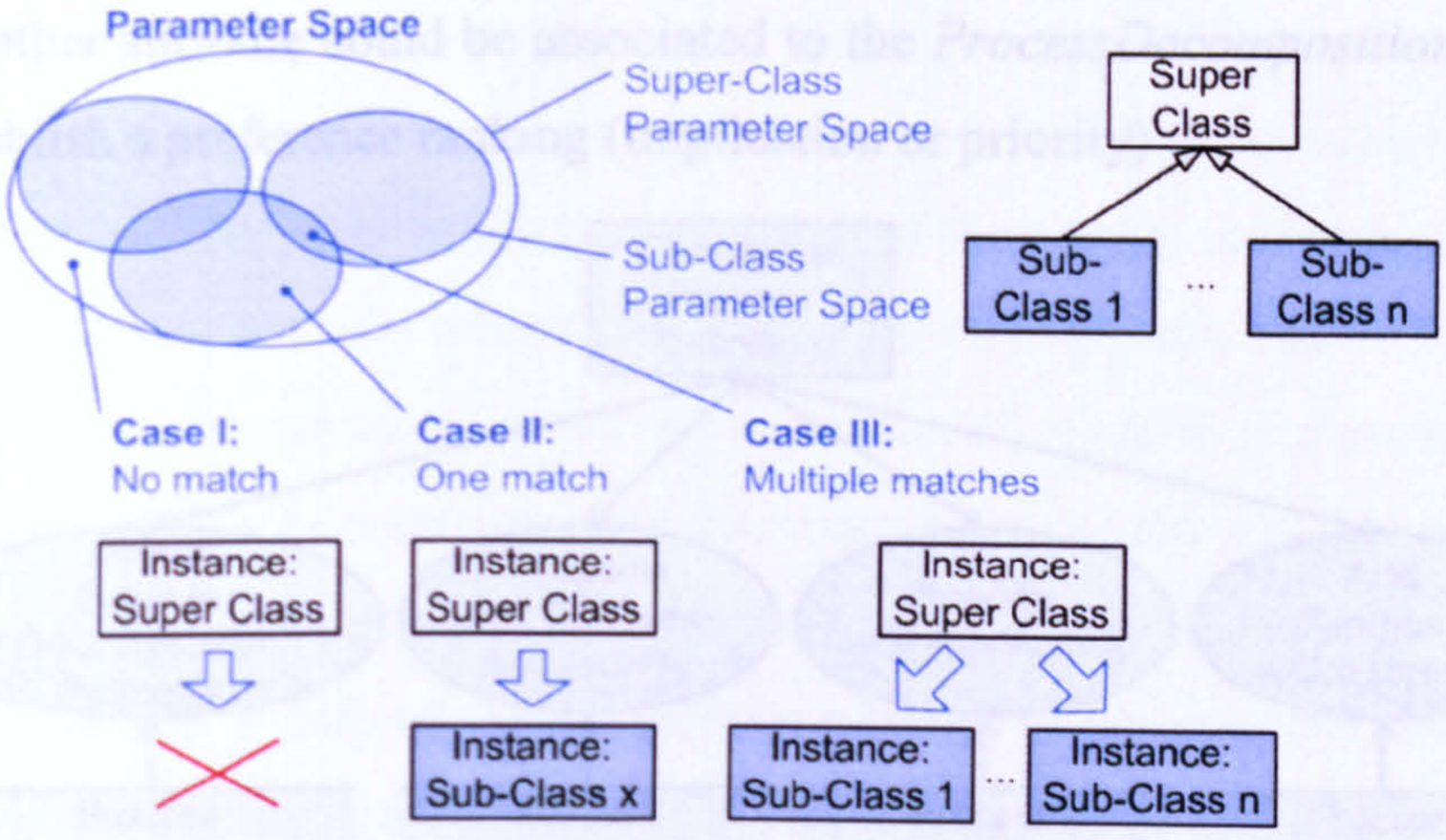


Figure 7.5 Principal cases of class specialisation

7.4.2 Process Decomposition Task

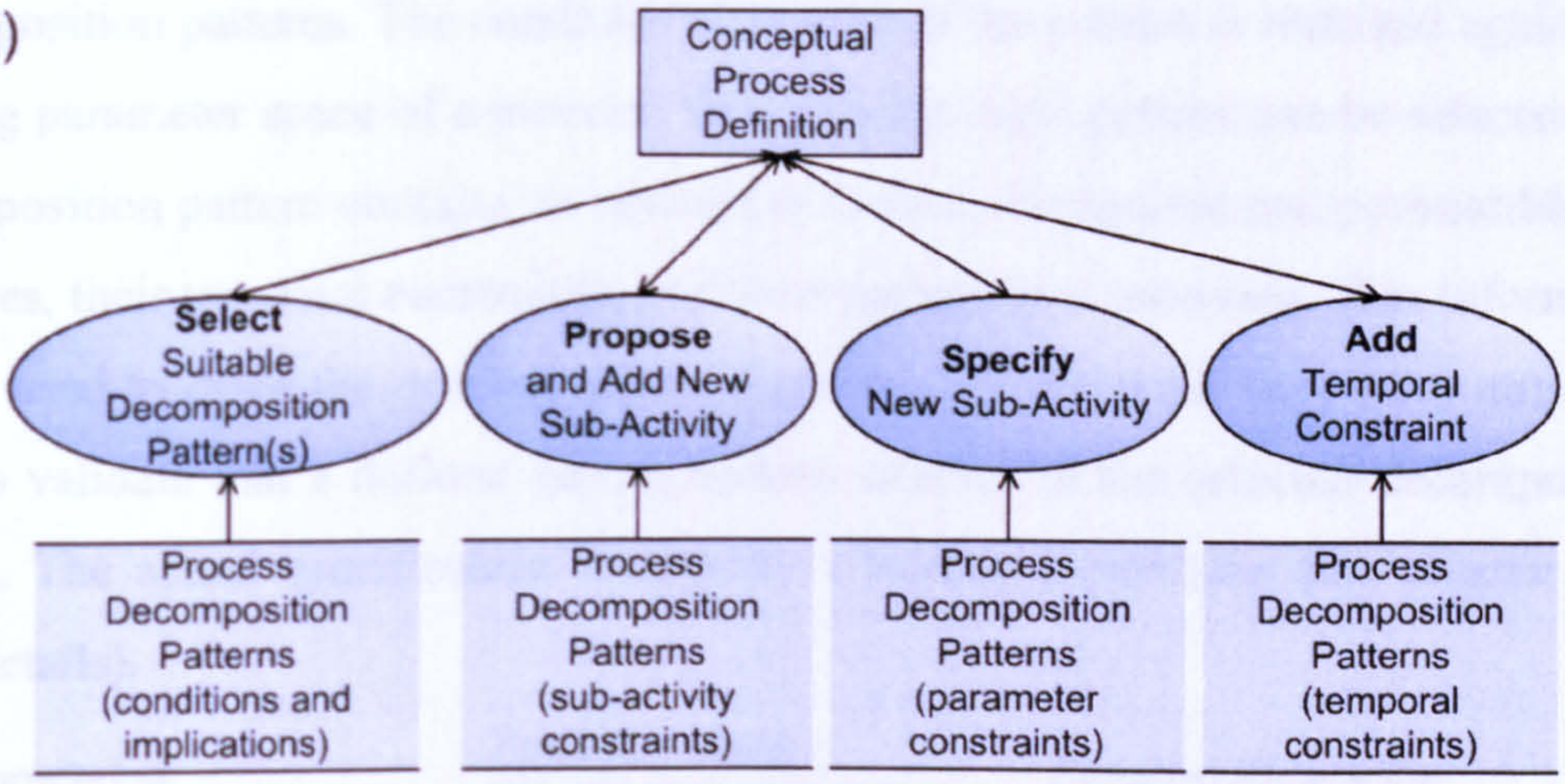
The process decomposition task is addressing the need to break down processes into more detailed specifications of their contained sub-activities. The task involves four activities: the selection of a suitable decomposition pattern; the adding of new sub-activities to the given process specification; the specification of the sub-activity parameters; and the adding of the temporal constraints for the new sub-activity (see Figure 7.6). All four activities use different aspects of the *ProcessDecompositionPatterns* (see also chapter 5.4.5). The *ProcessDecompositionPatterns* are associated to process types and any give process can be associated to more than one.

The *ProcessDecompositionPattern* contains a condition that defines for which parameter set it is applicable. This condition is used for the selection of a suitable *ProcessDecompositionPattern* for a given *Process* instance. Therefore, in principle the case could arise that the parameter set of a given *Process* instance matches more

than one condition. In this case the following strategies could be applied to overcome this indecisiveness:

1) This condition could be prevented by not allowing overlaps in the parameter space of the condition of one process type during the definition of the *ProcessDecompositionPatterns*
2) A human expert could be asked to intervene and select one of the choices
3) More than one decomposition could be included in the model depending on the choices with or without expert intervention
4) Another attribute could be associated to the *ProcessDecompositionPatterns* to establish a preference ranking (implication or priority)

a)



b)

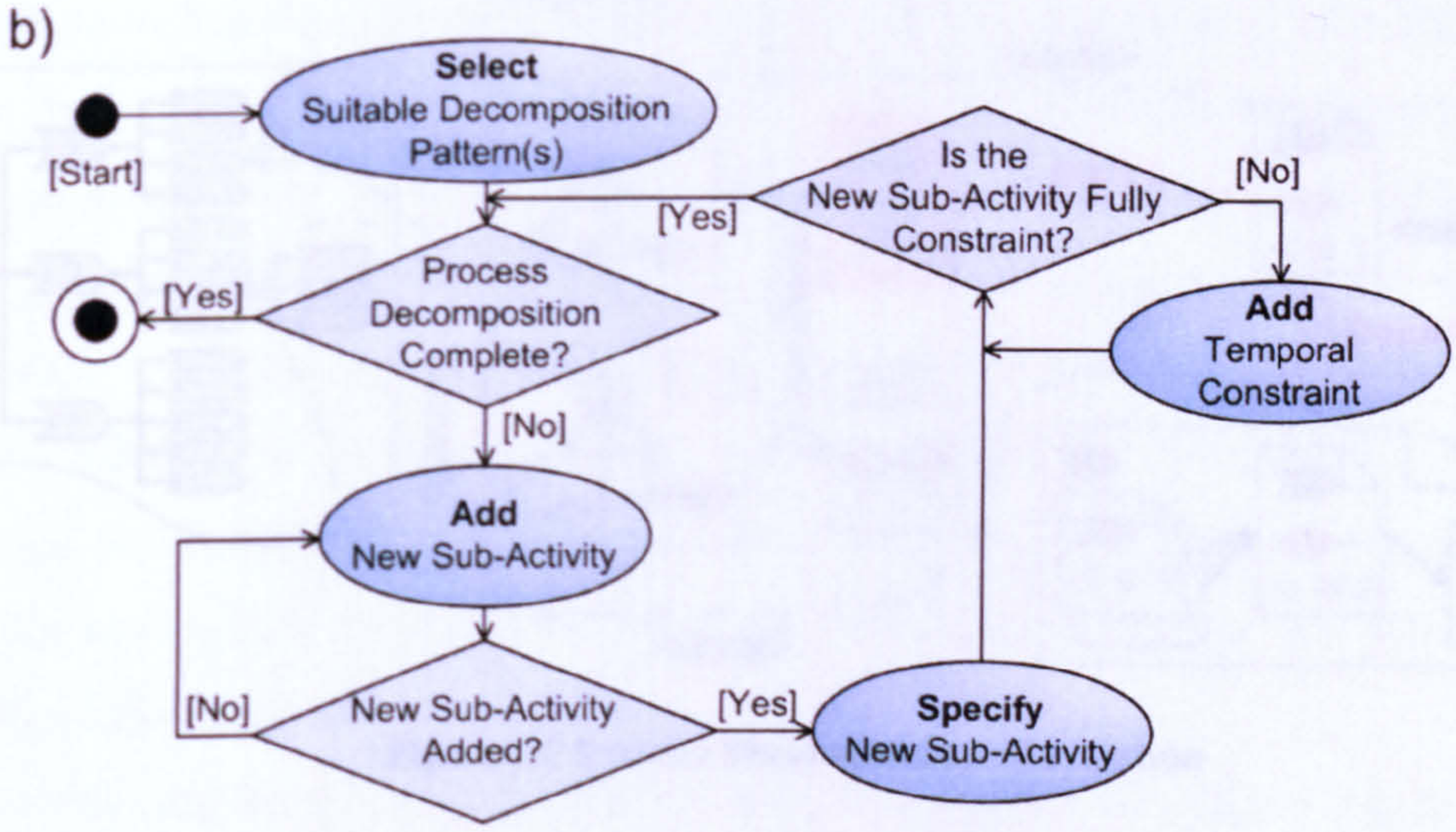


Figure 7.6 Process Decomposition Inference Structure a) knowledge transformation, b) control structure

For the current application it was assumed that option 2 will be used. A human expert can decide based on some textual description of the choices which

decomposition pattern to apply. Once a suitable pattern has been selected, it is associated to the process instance since the subsequent sub-activity specification activities only need to use the rules of this specific pattern.

After the selection of the decomposition pattern, its sub-activity constraints can be used to propose new sub-activities to be added to the process definition. Once a new sub-activity is added it can be specified based on the parameter constraints defined in the chosen *ProcessDecompositionPattern*. Finally, after the new sub-activity has been defined, its temporal constraints can be added to the process definition until it is fully constraint.

Figure 7.7 shows an illustration of the decomposition process based on process decomposition patterns. The condition statement of the pattern is matched against the existing parameter space of a process. This way the right pattern can be selected. The decomposition pattern contains an abstract definition of required and permissible sub-activities, their temporal constraints, and their parameter constraints. This information can be used to drive the decomposition. However, in this work they have only been used to validate that a defined decomposition adheres to the selected decomposition pattern. The actual specification is done by a human expert (see also chapter 8 for more details).

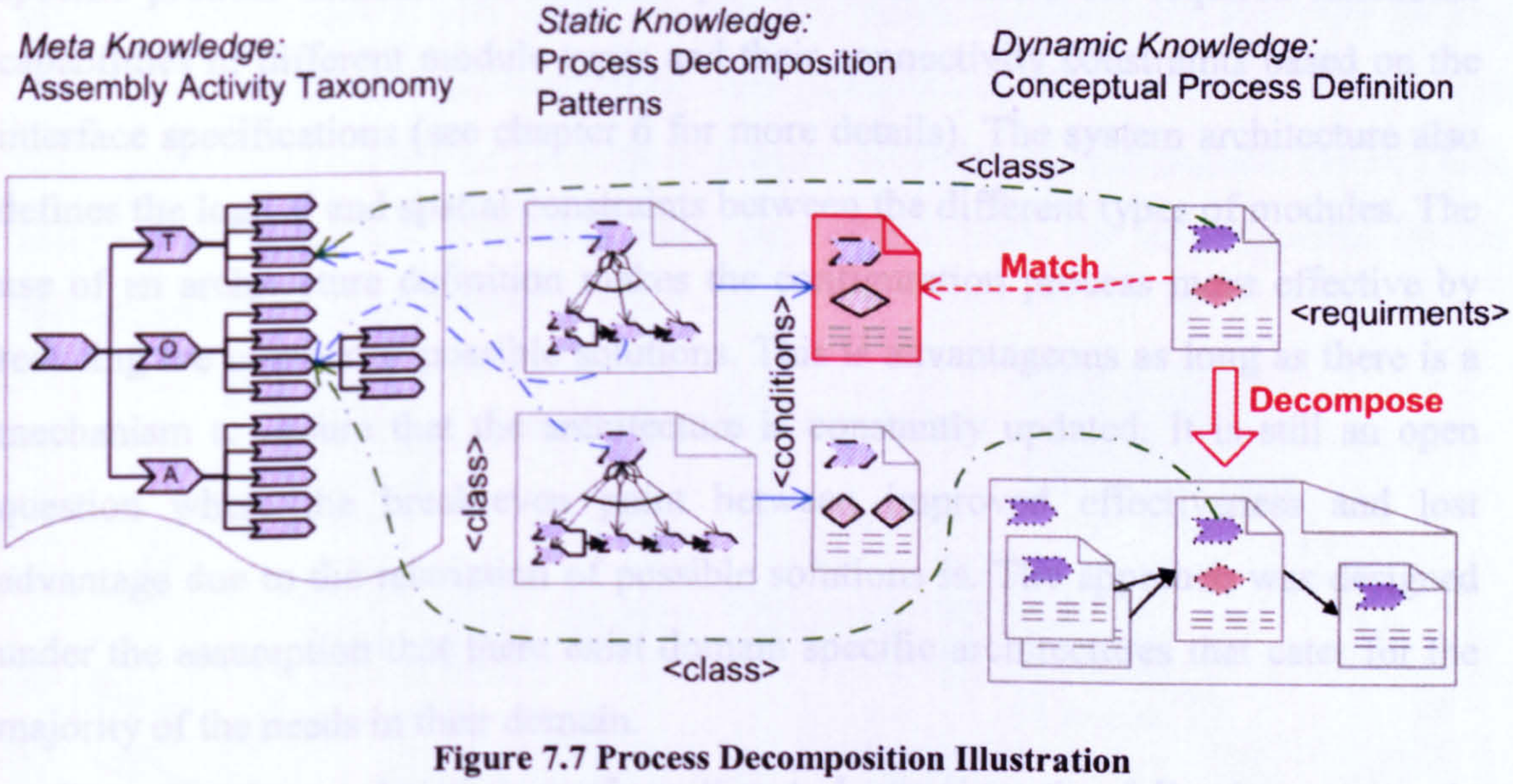


Figure 7.7 Process Decomposition Illustration

7.5 Requirements Driven Configuration of Modular Assembly Workstations

This section addresses the configuration of assembly workstation based on existing equipment modules. The equipment configuration is using a hierarchical configuration method that addresses the following aspects:

- grouping of activity requirements into conceptual equipment definitions
- specification of required equipment types and their specific requirements
- selection and evaluation of suitable equipment modules
- integration and functional synthesis of selected equipment modules
- evaluation of the performance characteristics of the integrated equipment modules

Each aspect of the configuration process is performed by domain experts. For example the selection and evaluation of equipment is made by a different expert for the different types of equipment. There is a domain expert that provides the capability to select and evaluate grippers, one to do the same for manipulators, etc.

The assembly equipment configuration is guided by predefined module types and interface specifications that are specified as part of a chosen system architecture for a specific product domain. The module specifications defined the required functional capabilities of different module types and their connectivity constraints based on the interface specifications (see chapter 6 for more details). The system architecture also defines the logical and spatial constraints between the different types of modules. The use of an architecture definition makes the configuration process more effective by reducing the number of possible solutions. This is advantageous as long as there is a mechanism to ensure that the architecture is constantly updated. It is still an open question where the break-even point between improved effectiveness and lost advantage due to the restriction of possible solutions is. The approach was designed under the assumption that there exist domain specific architectures that cater for the majority of the needs in their domain.

Generally the workstation configuration task requires the following two main inputs to propose new solutions:

- Workstation requirements are fully defined and not subject to change (static).
- There exists a library of equipment modules which are fully defined based on above definition and not subject to change (static).

Based on the given set of inputs the configuration task generates the following outputs:

- Workstation configuration(s) defined as a set of equipment modules and their interface connections.
- Additional processes required for the proposed workstation configuration to work.

In the following sections the specific design activities required during the configuration of modular assembly workstations are described in more detail. The description starts with a general overview of the fundamental approach followed by the individual design activities.

7.5.1 Hierarchical Approach

The equipment configuration process is generally following a problem decomposition and solution synthesis approach (Pahl and Beitz [94]). The overall problem definition of the assembly system is broken down into smaller sub-problems until a level has been reached where an existing equipment solution can be found. The existing solutions are evaluated and combined to form higher level solutions. The capabilities of the combined solutions are synthesised and compared with the original problem definitions at this level. Figure 7.8 shows a schematic overview of the assembly workstation decomposition and integration. The order in which the decomposition is carried out is only constraint between the hierarchical levels. This means that equipment units can only be defined after the requirements for its workstation have been defined.

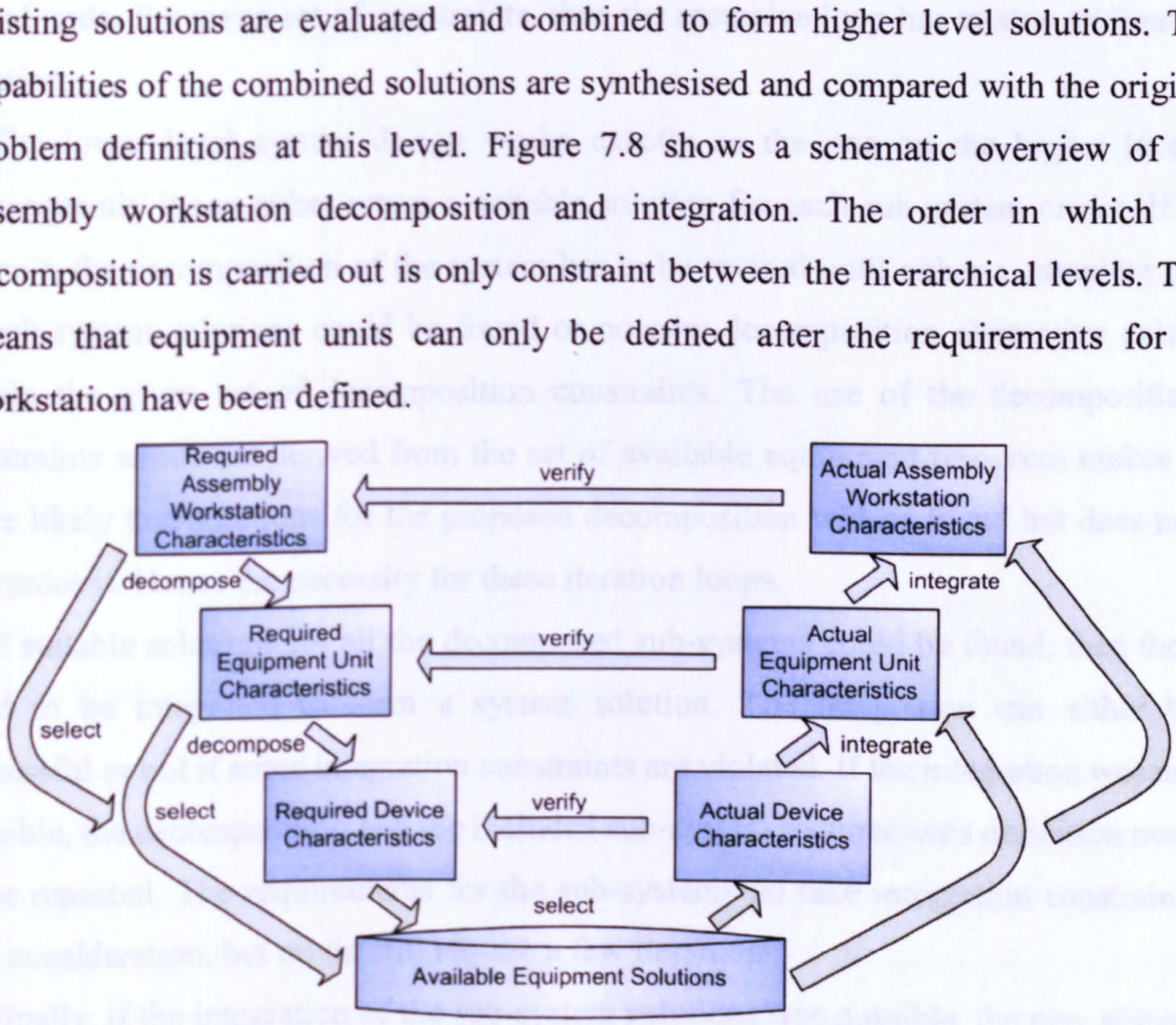


Figure 7.8 Hierarchical System Decomposition and Integration

Figure 7.9 shows a more detailed definition of the equipment configuration process. All the major design (knowledge transformation) tasks are shown as ovals in the diagram. The decisions that need to be taken at the end of each design task are shown as turned rectangles. More detail about the individual design tasks is going to be provided in the following sections.

The conceptual design (problem decomposition) and embodiment design (solution synthesis) take place in a recursive process. The overall design problem for an assembly system is defined through assembly process based system requirements. The design process starts on the highest level with a search for already existing solutions that can fulfil the given set of requirements. This is usually the system level. The design draws to a conclusion already if one or more exist and they are very likely to be an optimal solution. If not, the given problem definition is decomposed into sets of sub-problems either related to the sub-system (cell) or workstation level (see also equipment hierarchy). The next step is to find suitable solutions for all the sub-systems if a legal decomposition can be found. This starts the next recursive loop on the next lower level in the equipment hierarchy. If no legal decompositions can be found under the given set of constraints, then the recursive loop has to stop without a solution.

The lower level system design works exactly as the one on the higher level. Consequently it can either return a suitable solution for each sub-system or not. If it doesn't, the decomposition of the system has to be revised until either a complete set of sub-system solutions could be found or no new decomposition alternative exists within the given set of decomposition constraints. The use of the decomposition constraints which are derived from the set of available equipment resources makes it more likely that solutions for the proposed decomposition will be found but does not guarantee it. Hence the necessity for these iteration loops.

If suitable solutions for all the decomposed sub-systems could be found, then they need to be integrated to form a system solution. The integration can either be successful or not if some integration constraints are violated. If the integration was not possible, the decomposition and the included sub-system requirements definition need to be repeated. The requirements for the sub-systems do take integration constraints into consideration, but might still require a few iterations.

Finally, if the integration of the sub-system solutions was possible, the new system solution needs to be validated against the original system requirements on this level. If

the system fulfils the requirements it is proposed as a solution after checking how likely it is that there are still better solutions. If not, the decomposition is triggered again with additional constraints from the validation task.

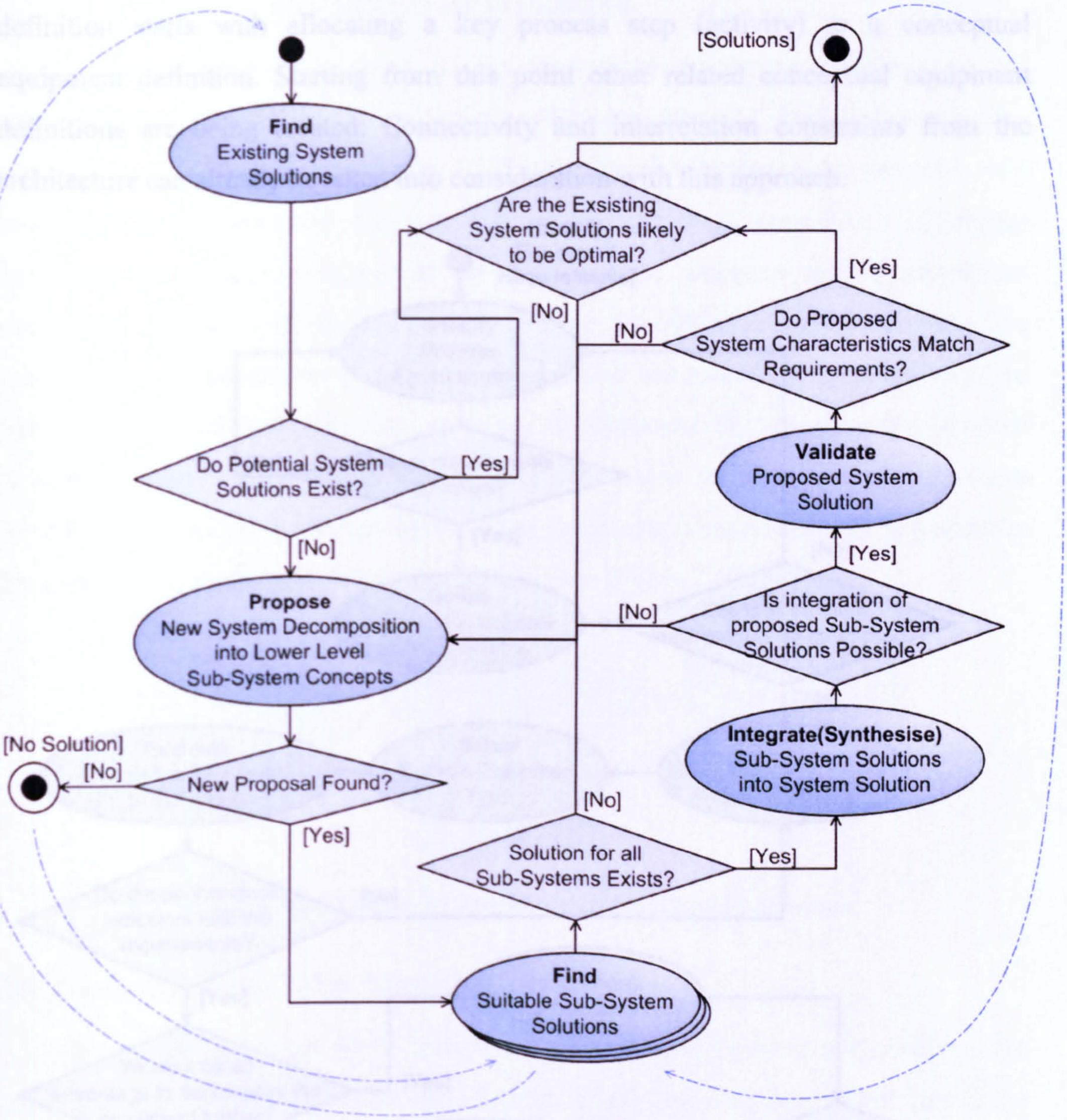


Figure 7.9 Recursive system decomposition and synthesis approach

7.5.2 Equipment Concept Decomposition Task

The system or problem decomposition task aims to split a given overall problem into next lower level sub-problems. The decomposition is based on existing solution concepts which are defined in the form of equipment type specifications (see chapter 6.4.7). They do not represent an individual piece of equipment but rather a class of equipment. The overall requirements are allocated to the different sub-problem definitions. For the decomposition of conceptual assembly equipment

specifications that means that the process requirements of the higher level equipment concept are grouped and allocated to the lower level ones. The allocation of process requirements and equipment type selection is an iterative process. In many cases the definition starts with allocating a key process step (activity) to a conceptual equipment definition. Starting from this point other related conceptual equipment definitions are being created. Connectivity and interrelation constraints from the architecture can already be taken into consideration with this approach.

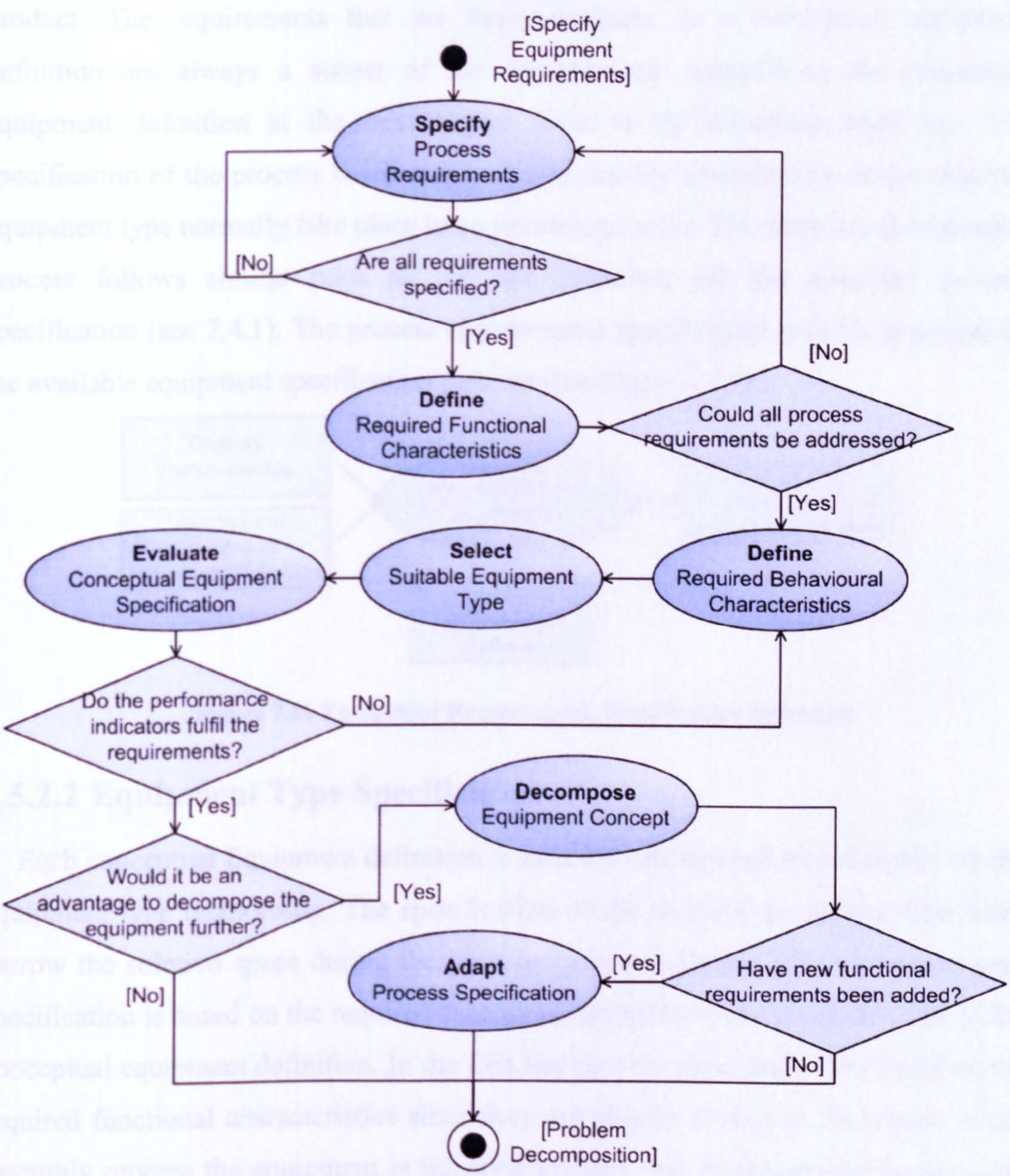


Figure 7.10 Conceptual Equipment Specification Overview

Once the decomposition of the overall conceptual equipment definition into lower level conceptual equipment definitions has been established the specific requirements

for each of the conceptual equipment can be defined. Finally the non-functional requirements are applied to each conceptual equipment definition.

7.5.2.1 Process Requirements Specification

The most fundamental aspect of the conceptual equipment definition is the establishment of the link to the assembly process requirements. They define what the equipment needs to be able to due to fulfil its assigned role during the assembly of a product. The requirements that are being assigned to a conceptual equipment definition are always a subset of the requirements assigned to the conceptual equipment definition at the next higher level in the structural hierarchy. The specification of the process based requirements and the specification of the required equipment type normally take place in an iterative process. This iterative specification process follows similar rules as the one described for the assembly process specification (see 7.4.1). The process requirements specification activity is guided by the available equipment specification patterns (see Figure 7.11).

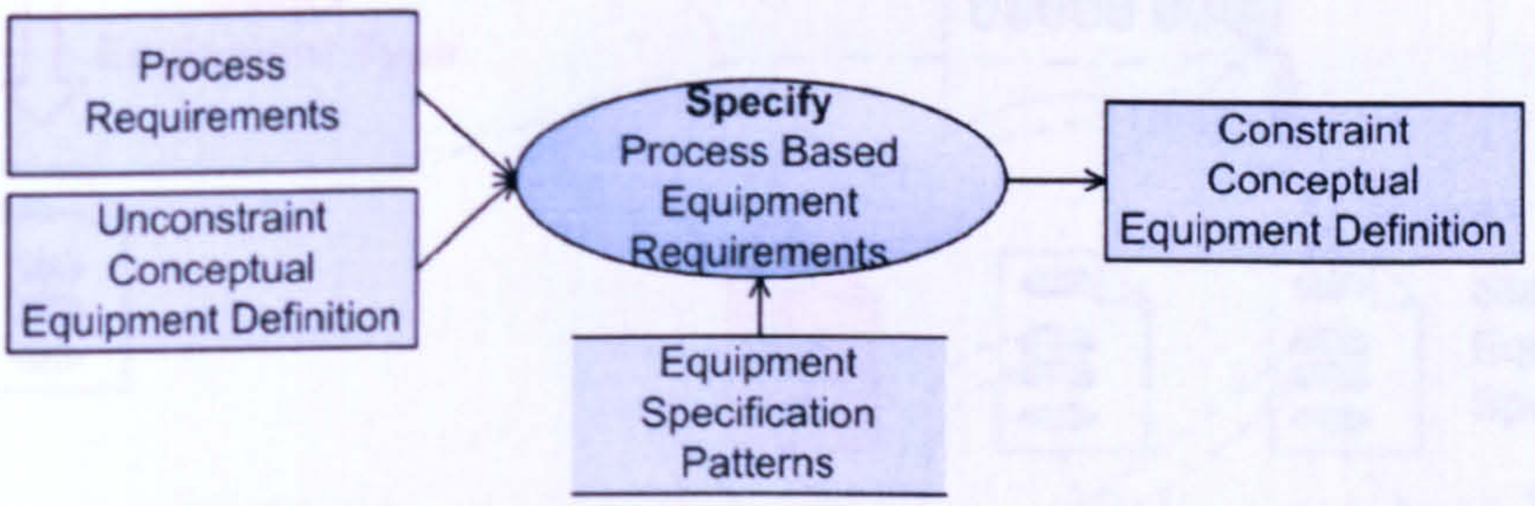


Figure 7.11 Equipment Requirements Specification Inference

7.5.2.2 Equipment Type Specification

Each conceptual Equipment definition is linked to the equipment taxonomy via the equipment type relationship. The specification of the required equipment type helps narrow the solution space during the later equipment selection. The equipment type specification is based on the required functional and behavioural characteristics of the conceptual equipment definition. In the first instance the classification is based on the required functional characteristics since they are closely related to the aspect of the assembly process the equipment is intended to carry out. In the second instance the equipment type is related to the more specific behaviour that is found to be advantageous for the given assembly problem.

Figure 7.12 shows a schematic overview of the equipment type specification inference. A conceptual equipment definition including at least some high level

process requirements needs to be given as input knowledge. The static knowledge used to guide the specification is given through equipment specification patterns (see chapter 6 for more details). The inference modifies the equipment type attribute of the input knowledge and makes it more specific.

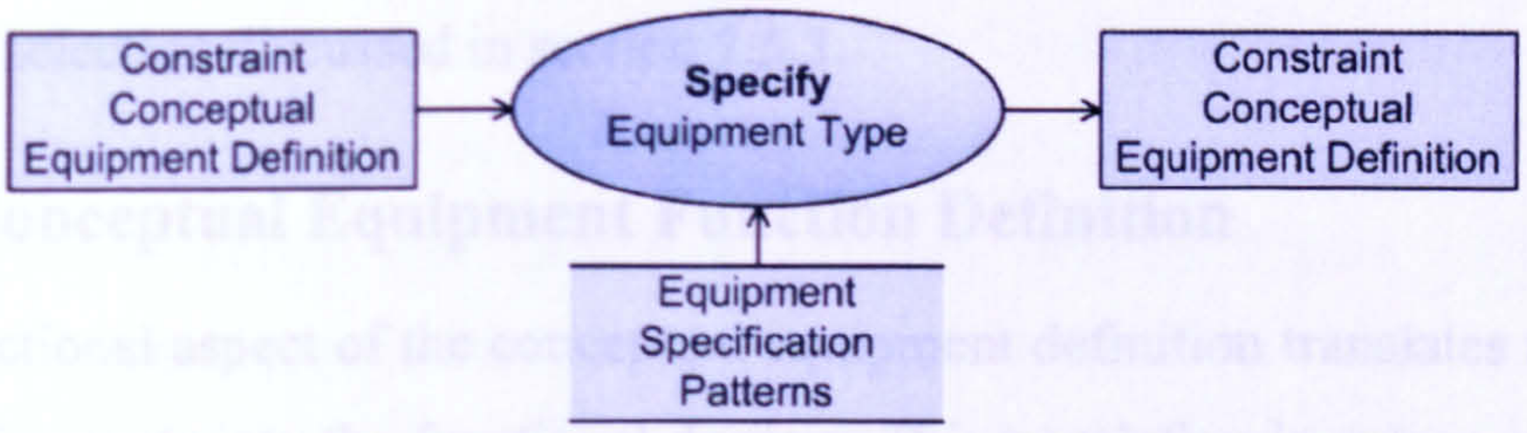


Figure 7.12 Conceptual Equipment Type Specification Inference

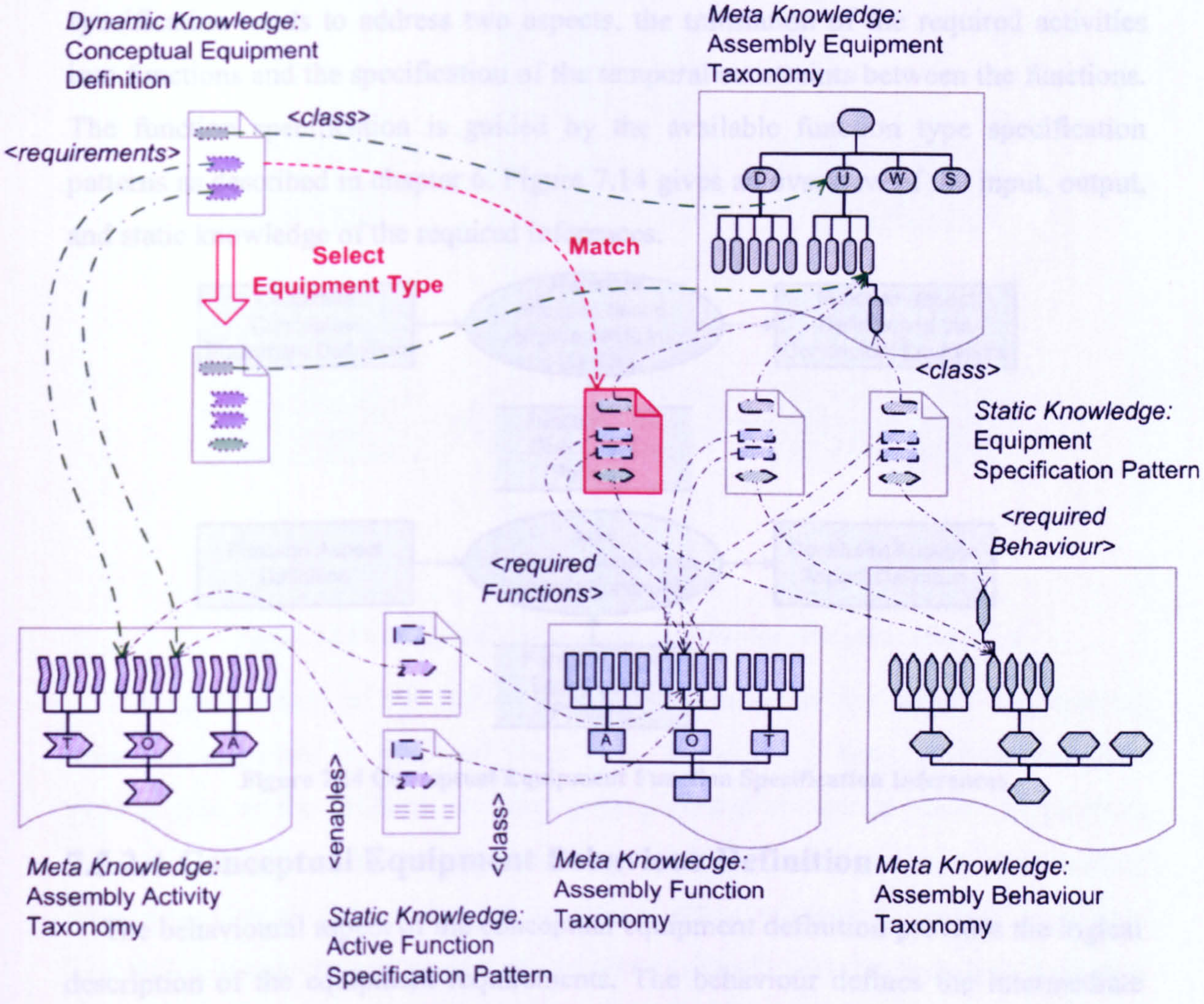


Figure 7.13 Illustrative Equipment Type Specialisation

Figure 7.13 shows an illustration of the equipment type specification activity. The selection of the right type takes place in iteration with the specification of the required functional and behavioural characteristics. This is the case since both the required

functional and behavioural characteristics of an equipment influence which type is required. This approach reflects the gradual transition from more abstract functional requirements over more specific behavioural specification to the final choice of a suitable embodiment (Rosenman and Gero [107]). The last step is part of the equipment selection discussed in section 7.5.3.

7.5.2.3 Conceptual Equipment Function Definition

The functional aspect of the conceptual equipment definition translates the process based requirements into the functional domain. This translation is only required if the functional definition space is different from the process space. The function specification needs to address two aspects, the translation of the required activities into functions and the specification of the temporal constraints between the functions. The function specification is guided by the available function type specification patterns as described in chapter 6. Figure 7.14 gives an overview of the input, output, and static knowledge of the required inferences.

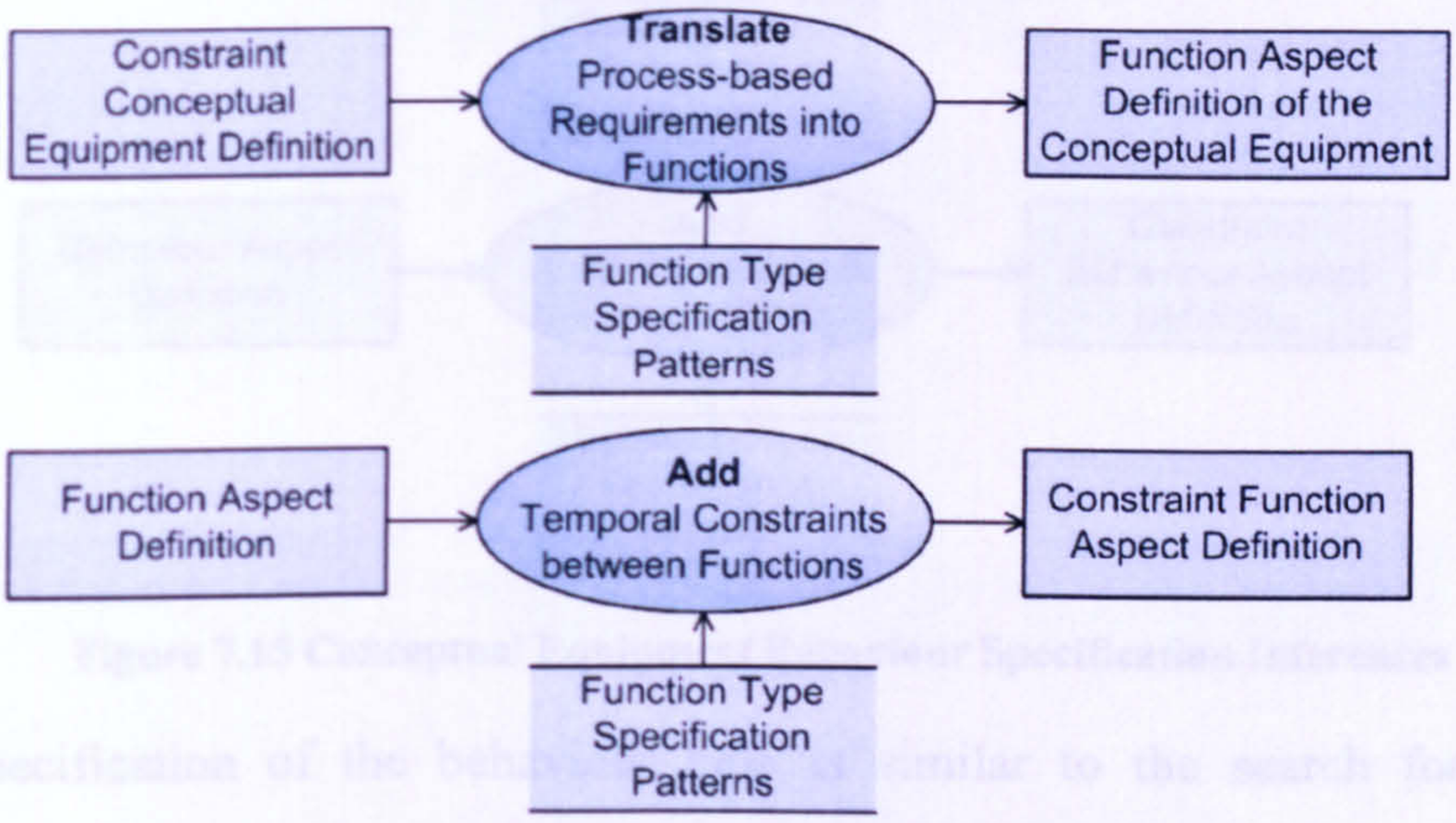


Figure 7.14 Conceptual Equipment Function Specification Inferences

7.5.2.4 Conceptual Equipment Behaviour Definition

The behavioural aspect of the conceptual equipment definition provides the logical description of the equipment requirements. The behaviour defines the intermediate level between the very abstract functional description and the specifics of the structural description. The behaviour specification is in the first instance based on the functional description of the conceptual equipment entity in question. Some alterations might later be required due to some structural restrictions.

The specification of the behaviour needs to address the following three aspects. First a mechanism needs to be provided to add new behaviour definitions and relate them to the functions they enable. Secondly there needs to be a mechanism that allows the type of already defined behaviours to be defined or revised. Finally the logical relationships between the required behavioural characteristics need to be added. The behaviour type specification patterns regulate and constrain these three aspects of the definition (see chapter 6 for more details). Figure 7.15 shows an overview of the three required definition inferences.

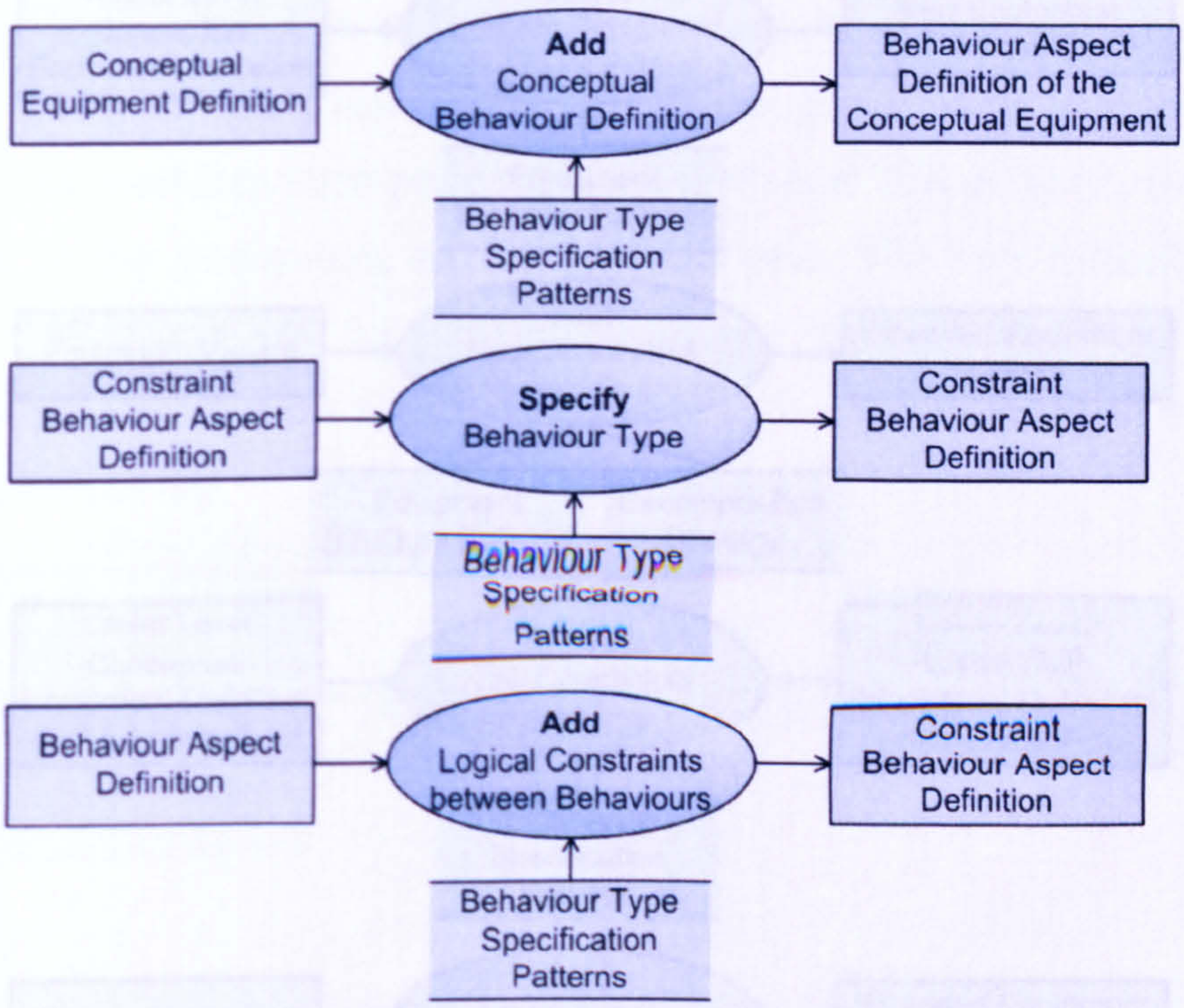


Figure 7.15 Conceptual Equipment Behaviour Specification Inferences

The specification of the behaviour type is similar to the search for working principles described by Pahl and Beitz [94]. The different types of behaviours are synonymous to the different working principles and should be based on physical phenomena at the lowest level (Pahl and Beitz [94]). However, this very high level of detail is not required for the requirements definition of modular assembly systems. The choice of appropriate physical principles has already been made during the design of the available equipment modules. Only the behaviour constraints arising from them need to be considered at this point.

7.5.2.5 Conceptual Equipment Structure Definition

The structure specification during the conceptual equipment definition is the main mechanism for the design problem decomposition at the equipment side. The equipment concept decomposition is following a very similar approach as the process decomposition described in section 7.4.2. The structural decomposition is defining the hierarchical structure of the required system solution. This is due to the fact that the solution will be composed from physically independent equipment modules.

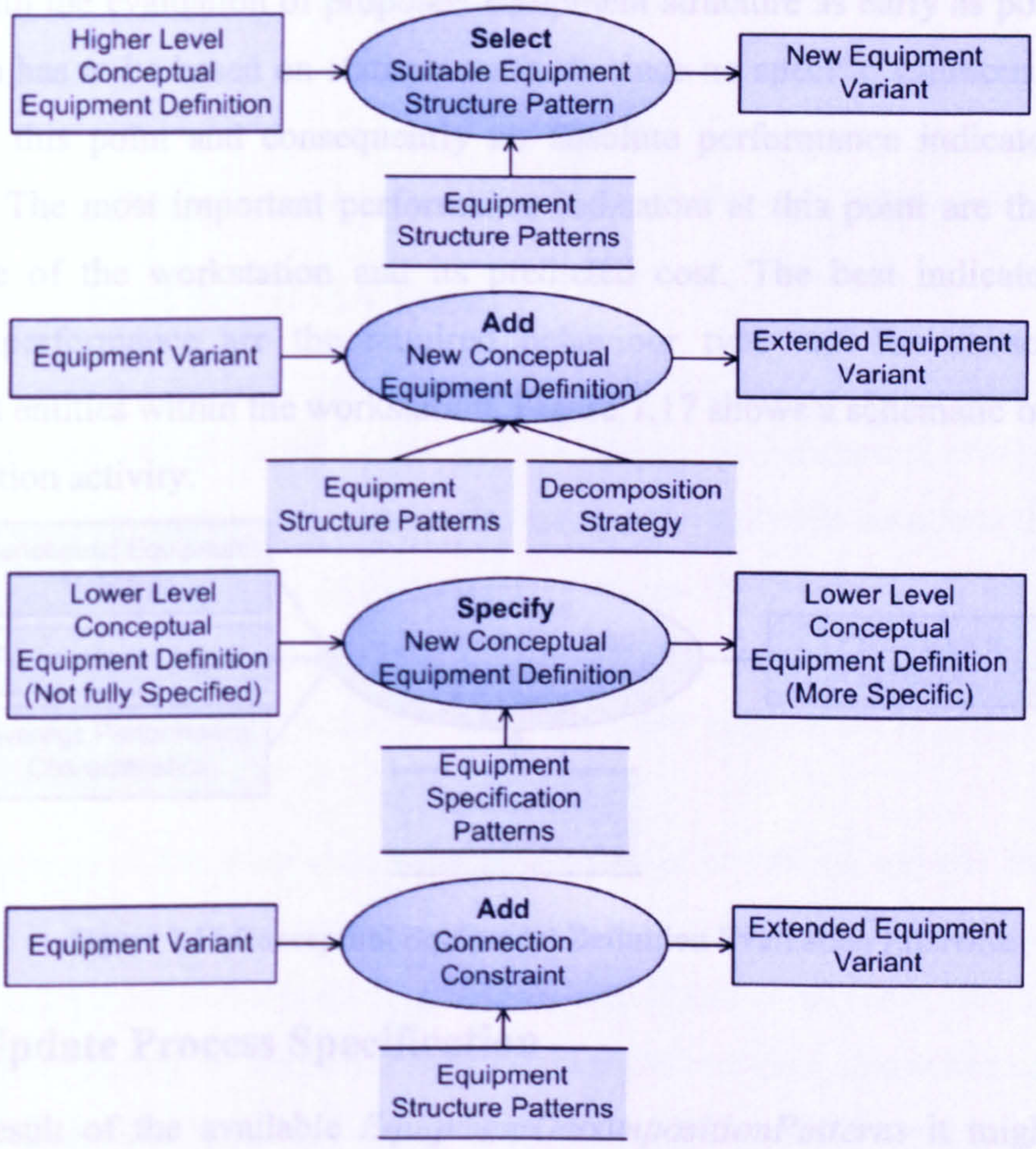


Figure 7.16 Conceptual Equipment Structure Specification Inferences

The structure definition should address four points. First, a suitable structure pattern needs to be chosen. This will not be a fixed decision but rather should be an iteration between the choices of lower level conceptual equipment definitions and the choice of structure pattern. Secondly, new lower level conceptual equipment definitions should be added according to the selected structure pattern. Thirdly, the requirements, functional, behavioural, and structural aspects of the new conceptual equipment definition need to be specified in accordance with the steps described above. Finally, the connection constraints between the lower level conceptual

equipment definitions need to be added once at least two have been sufficiently specified. Figure 7.16 shows an overview of the input, output, and static knowledge involved in the design activities.

7.5.2.6 Evaluation of Conceptual Equipment Definition

The decomposition and specification of the conceptual equipment module is further restricting the solution space for the assembly system. It is therefore important to start with the evaluation of proposed equipment structure as early as possible. The evaluation has to be based on statistic methods since no specific solutions have been chosen at this point and consequently no absolute performance indicators are yet available. The most important performance indicators at this point are the expected cycle type of the workstation and its predicted cost. The best indicators for the expected performance are the required behaviour type and the chosen type of equipment entities within the workstation. Figure 7.17 shows a schematic overview of the evaluation activity.

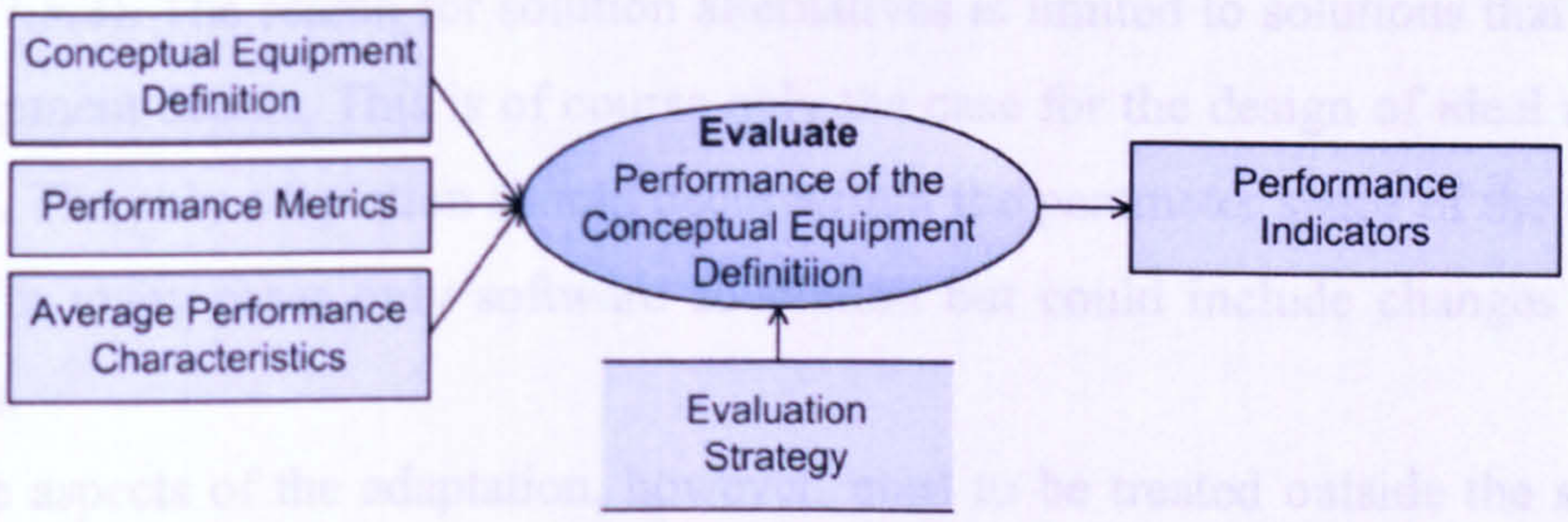


Figure 7.17 Conceptual Equipment Definition Evaluation Inference

7.5.2.7 Update Process Specification

As a result of the available *EquipmentDecompositionPatterns* it might become necessary to add supporting equipment concepts. They normally have functions of their own. Consequently, the execution of their functions needs to be added to the process definition to maintain the logical and temporal integrity between the process and the equipment specification. The design activity takes place as part of the assembly process domain. The equipment specification tasks only request new activities to be added based on the functional needs of the new equipment concept. The new functional aspects of the model need to be translated back into activities before they can be added to the process specification. Figure 7.18 shows a schematic overview of the design activities required to update the process specification.

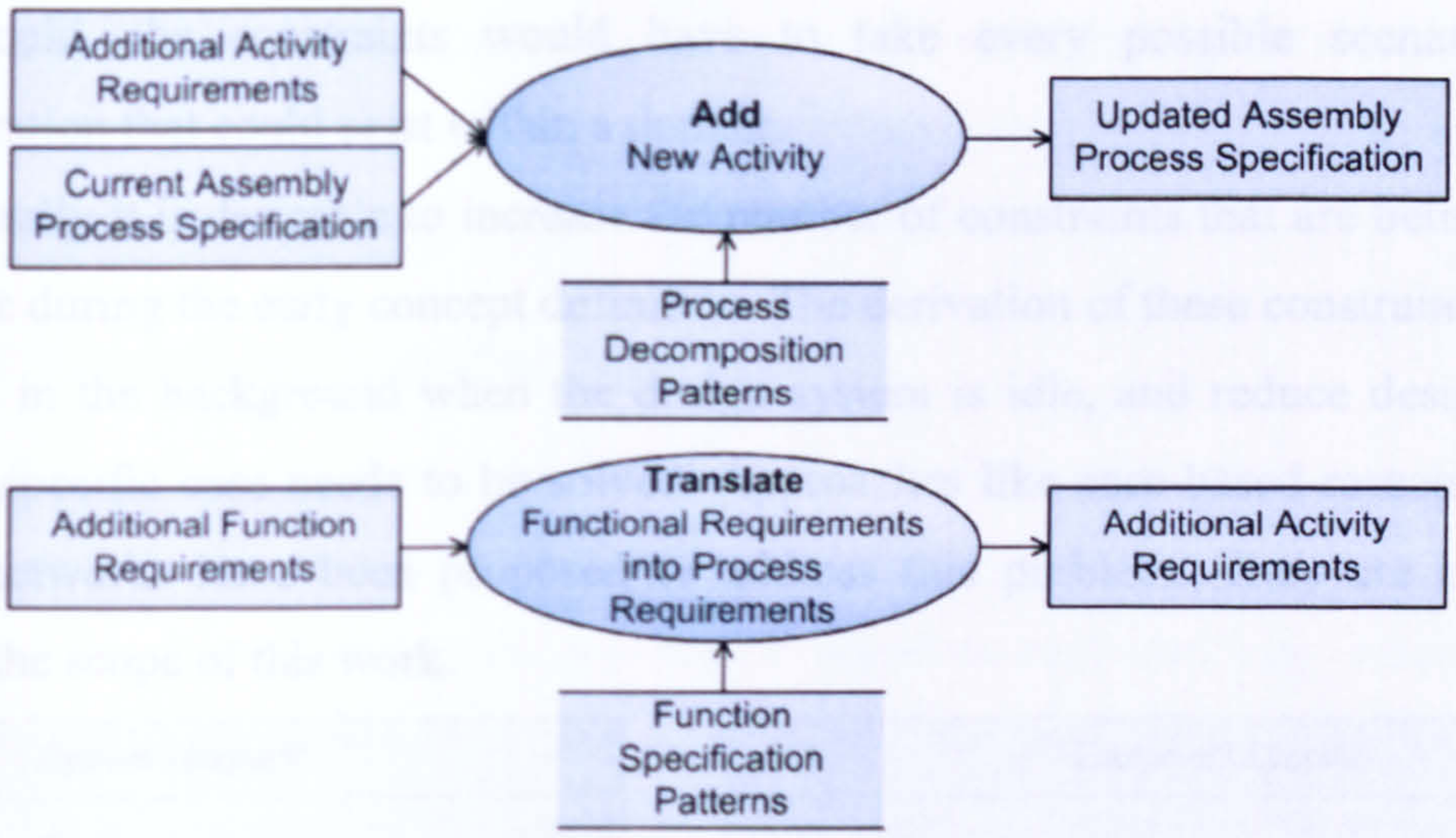


Figure 7.18 Update Assembly Process Specification Inference

7.5.3 Equipment Selection Task

The equipment selection task is addressing the need to find existing equipment solutions for the sets of requirements defined during the conceptual design task (see section 7.5.2). The search for solution alternatives is limited to solutions that exist in the equipment depots. This is of course only the case for the design of ideal modular systems. The only adaptation should occur within the parameter space of the module. This is in many cases only software adaptation but could include changes of sub-modules.

Some aspects of the adaptation, however, need to be treated outside the scope of the modular system architecture. The most obvious case is the adaptation of functional equipment to the specific geometric conditions of the component parts they need to handle during an assembly process. The close-to-part adaptation has to be in many cases individually defined which is the reason it has to be outside the scope of the module architecture. However, the system architecture should make this adaptation as easy as possible. For example the fingers of a mechanical gripping tool can be easily replaced to adapt the gripper to new part geometries. The basic functionality of the device does not change so.

The requirements definition during the conceptual design is already taking the constraints arising from the available equipment solutions into consideration. This makes it much likelier that solutions can be found. However, the design constraints used during the conceptual design are an abstraction of the available equipment capabilities. They cannot guarantee that a solution exists but make it more likely. If

they would, the constraints would have to take every possible scenario into consideration that could exist within a domain.

Generally it is desirable to increase the number of constraints that are being made available during the early concept definition. The derivation of these constraints could be done in the background when the design system is idle, and reduce design time when a specific case needs to be solved. Approaches like case-based-reasoning and neural networks have been proposed to address this problem. They are however outside the scope of this work.

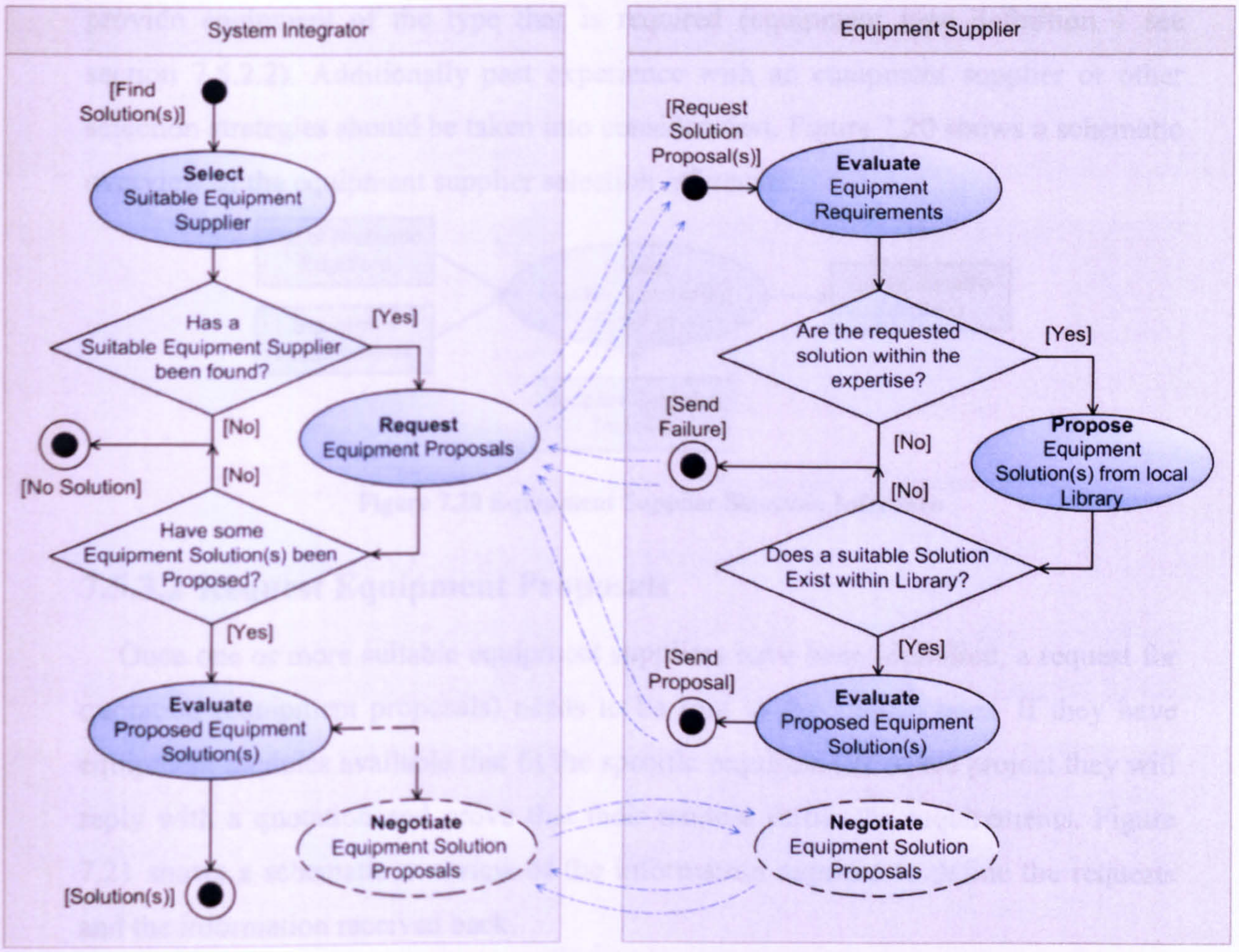


Figure 7.19 Find Existing Equipment Solutions

The selection of suitable equipment solutions is in many cases domain or even equipment type specific. The selection approach consequently has to allow for different selection strategies to be applied. Furthermore, the selection and adaptation of suitable equipment solutions is often done by someone other than the system integrator. The equipment selection approach has been split into two parts to address those needs: a system integrator side (see Figure 7.19 left side) and an equipment supplier side (see Figure 7.19 right side).

The system integrator should only have to deal with finding suitable equipment suppliers who can provide equipment solutions that meet the specific requirements of the equipment concepts needed for the overall function of the system.

7.5.3.1 Equipment Supplier Selection

The first step during the search for suitable equipment solutions in a multi-vendor environment is the selection of suitable equipment providers. This is done by searching the list of registered equipment suppliers and checking whether they provide equipment of the type that is required (equipment type definition – see section 7.5.2.2). Additionally past experience with an equipment supplier or other selection strategies should be taken into consideration. Figure 7.20 shows a schematic overview of the equipment supplier selection inference.

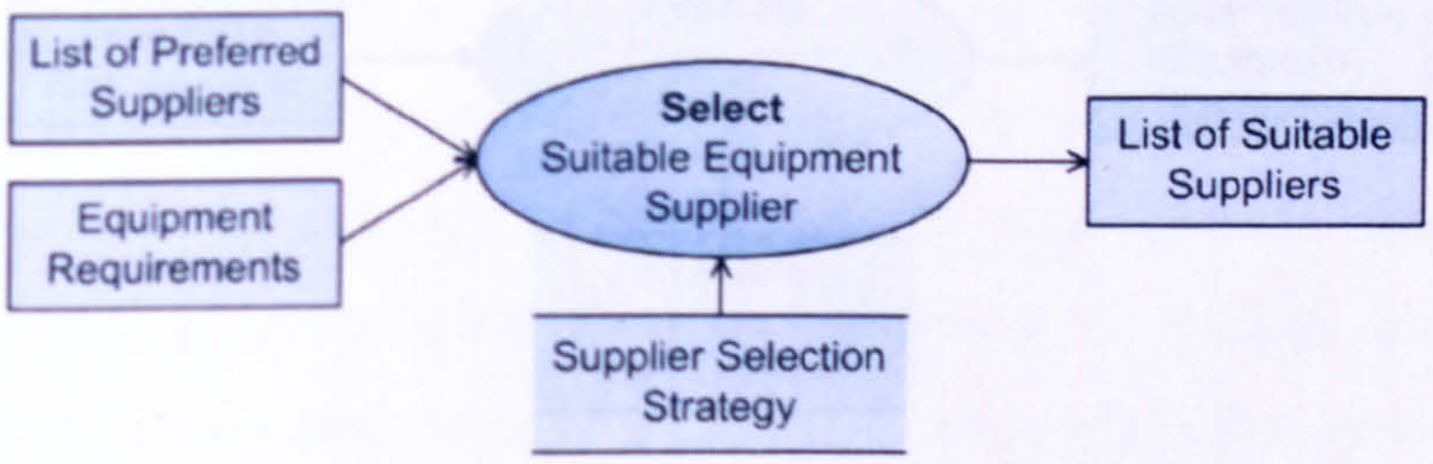


Figure 7.20 Equipment Supplier Selection Inference

7.5.3.2 Request Equipment Proposals

Once one or more suitable equipment suppliers have been identified, a request for quotation (equipment proposals) needs to be sent to the chosen ones. If they have equipment modules available that fit the specific requirements of the project they will reply with a quotation and prove that their module fulfils the requirements. Figure 7.21 shows a schematic overview of the information required to define the requests and the information received back.

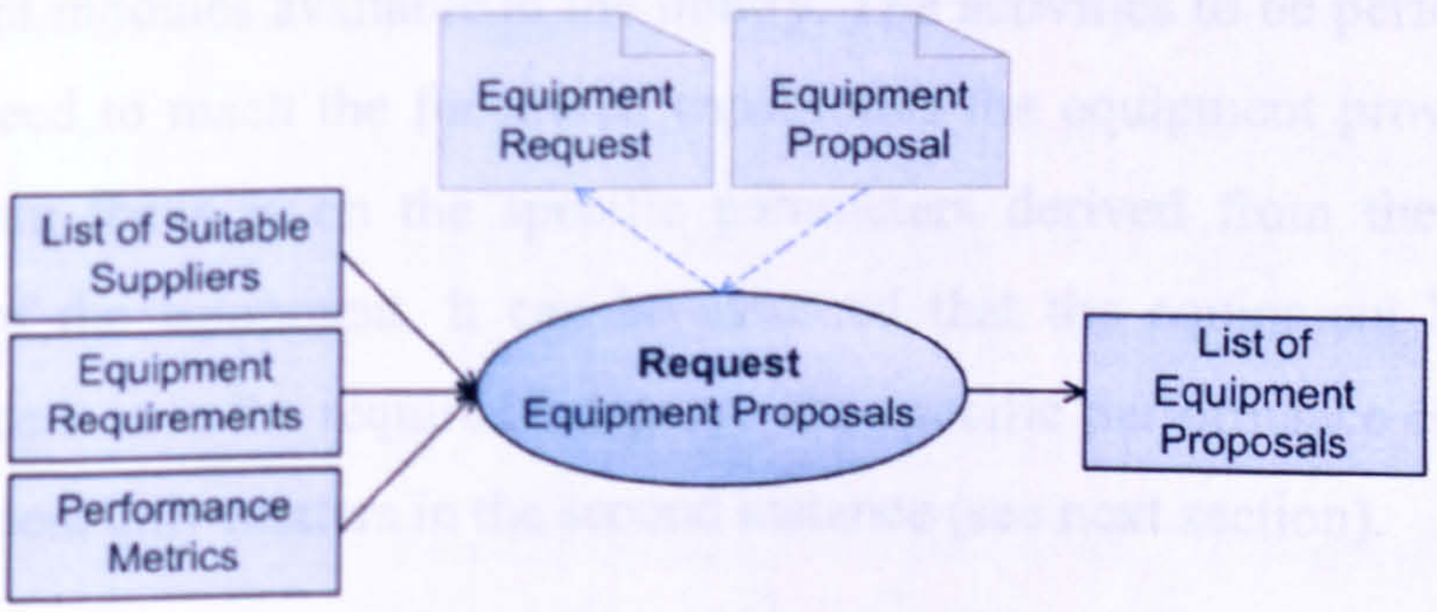


Figure 7.21 Request Equipment Proposals Inference

7.5.3.3 Evaluate Equipment Requirements

The actual equipment selection in a distributed environment could be done by someone other than the originator of the equipment requirements which need to be fulfilled. This requires whoever is providing this kind of service to make sure that the requirements for the requested proposal are actually within his area of expertise. Simultaneously it is often required to narrow down the type of equipment that is needed, based on the specific knowledge of the equipment supplier. Figure 7.22 shows an overview of the knowledge transformation that takes place during the equipment requirements evaluation. The required equipment category or type is in the first instance determined by the functions required and only in the second by the different behaviours used to achieve them.

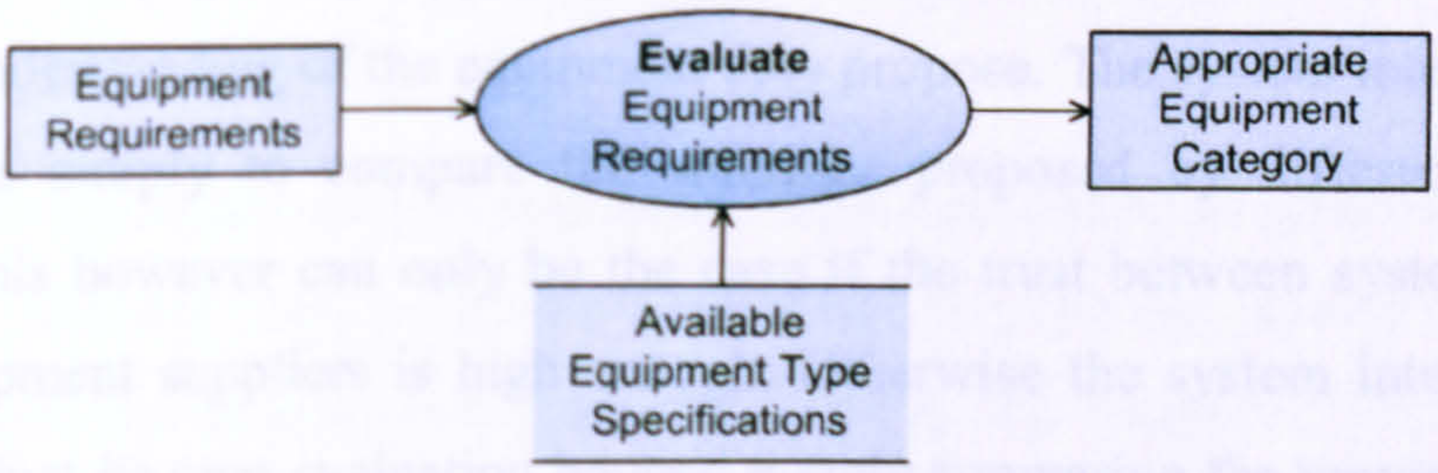


Figure 7.22 Evaluate Equipment Requirements Inference

7.5.3.4 Equipment Selection from Library

Once it has been determined that the requirements of the requested equipment solution are within the equipment supplier's area of expertise, he can proceed to the more effort intensive task of finding specific solutions that fit the requirements. The selection of equipment solutions primarily needs to determine that the technical capability of the equipment meets the requested requirements (see Figure 7.23). The selection of modular equipment solutions is driven by the functional capabilities of the equipment modules available in the library. The activities to be performed by the equipment need to mach the functional capabilities the equipment provides. At this level the main focus is on the specific parameters derived from the behavioural description of the equipment. It can be assumed that the equipment has the right functions once it is in the required category. The specific performance characteristics of the equipment only matters in the second instance (see next section).

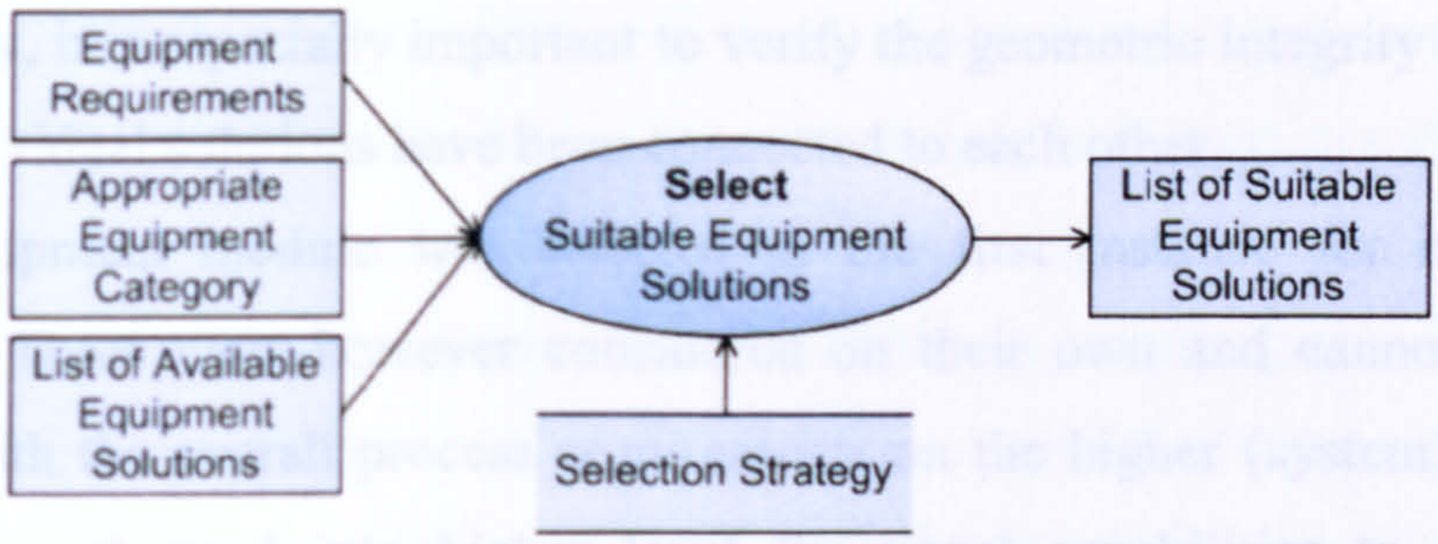


Figure 7.23 Select Equipment Solutions Inference

7.5.3.5 Equipment Evaluation

The evaluation of equipment solutions can take place both on the equipment supplier side and on the system integrator side. If suitable performance metrics are supplier to all equipment suppliers, they can do the evaluation based on their additional understanding of the equipment they propose. The system integrator would then be able simply to compare the solutions proposed by different equipment suppliers. This however can only be the case if the trust between system integrator and all equipment suppliers is high enough. Otherwise the system integrator might want to conduct its own evaluation beyond merely comparing the provided rankings. Figure 7.24 shows a schematic overview of the information transformation taking place during the equipment solution evaluation independent of who conducts it.

Figure 7.24 Equipment Solution Evaluation Inference

7.5.4 Assembly Cell Configuration Task

Once individual equipment solutions for the lower level conceptual design definitions have been found, they need to be integrated into a higher level functional solution. The first step is to establish that the proposed equipment solutions actually all fit together on the physical, energy, and information levels. Some connectivity constraints are already being taken into consideration during the conceptual design of the system. However, some equipment specific conflicts particularly on the geometric level cannot be foreseen at the more abstract level of the design constraints.

Consequently, it is especially important to verify the geometric integrity of the design once the individual solutions have been connected to each other.

Each equipment module was selected in the first instance for its functional capabilities. These were however considered on their own and cannot directly be compared with the overall process requirements on the higher (system) level. They need to be synthesised into higher level functional capabilities to perform this validation.

Different strategies could be used to configure a set of equipment modules. A likely strategy for the decomposition and subsequent integration could be to start with the most critical piece of equipment and arrange all the others around it. However, the choice of the approach is up to the design expert.

7.5.4.1 Equipment Integration

The actual equipment module definitions can be connected as soon as two or more have been selected for a given conceptual equipment specification. Their interconnection has three aspects: physical, logical, and temporal. These reflect the structure, behaviour, and function aspects of the equipment definition. The physical and logical connection is treated in this section because they use a very similar mechanism to guide their specification. The definition of the temporal relationships between the functions of the modules will be discussed together with the functional synthesis in section 7.5.4.2 below.

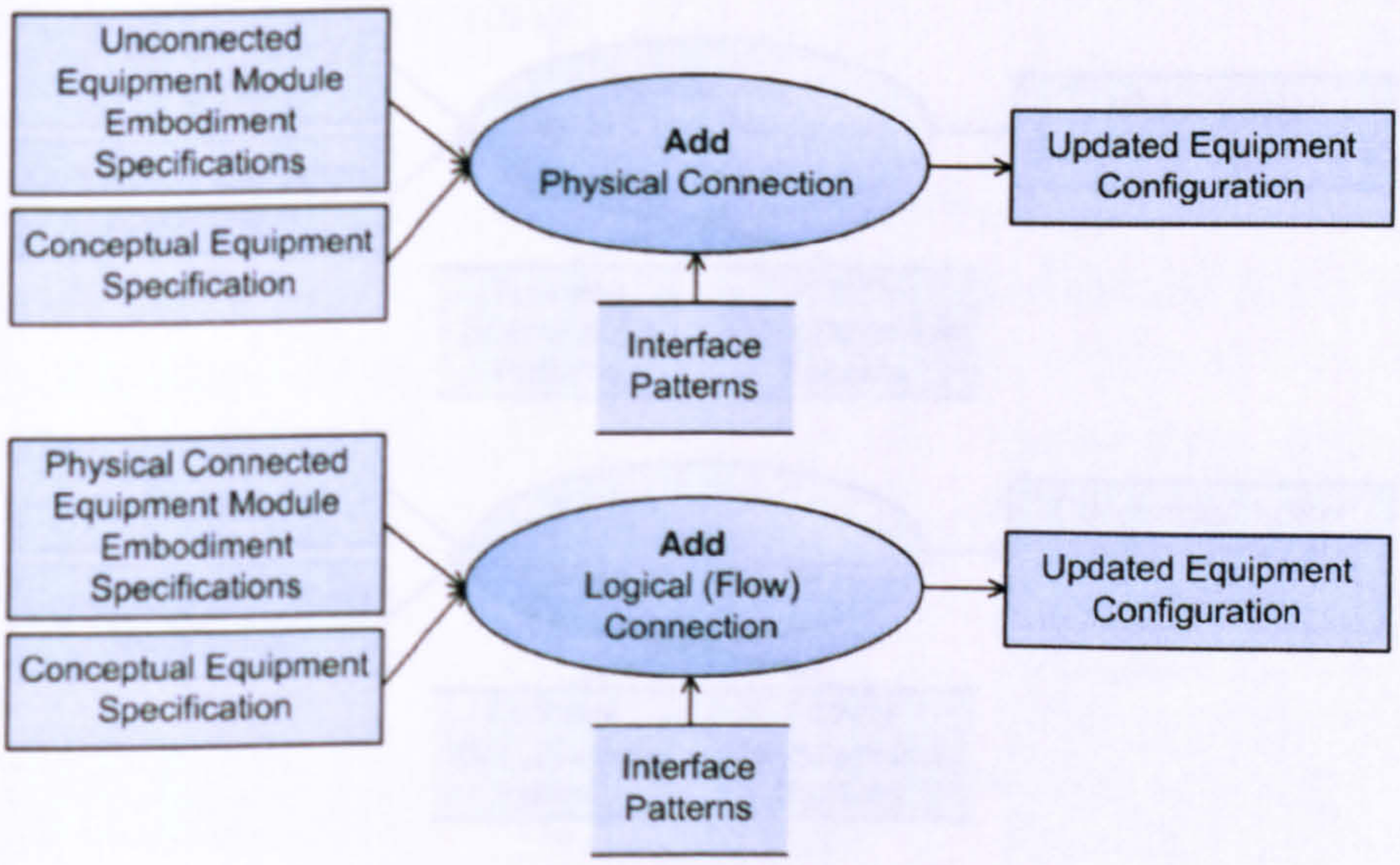


Figure 7.25 Equipment Integration Inferences

Figure 7.25 shows a schematic overview of the two design activities. Both activities have the specifications of the two actual equipment modules that need to be connected as an input. Additionally they have the original conceptual equipment specification as an input to indicate how the two should be connected. The *InterfacePatterns* (see chapter 6.4.8) define the characteristics of the connection.

Theoretically, these two design activities could take place in parallel. However, generally the physical connections are defined first which reflects the notion that only connected modules can have logical interrelationships. This also reflects the general design approach which is defined as the reverse process of the decomposition. Consequently, the physical connection has to be first since the physical specification was the last step of the decomposition.

7.5.4.2 Functional Synthesis

The functional synthesis activity during the integration of selected equipment embodiment specifications addresses the need to understand and validate the functional capabilities of the connected equipment modules. This is important for two reasons. Firstly, the synthesis is needed to verify that the combined functional capabilities of equipment modules that were selected at a lower level actually still match the original requirements. Secondly, it is important particularly for the reconfiguration of existing equipment to understand also the unplanned functional capabilities of the equipment.

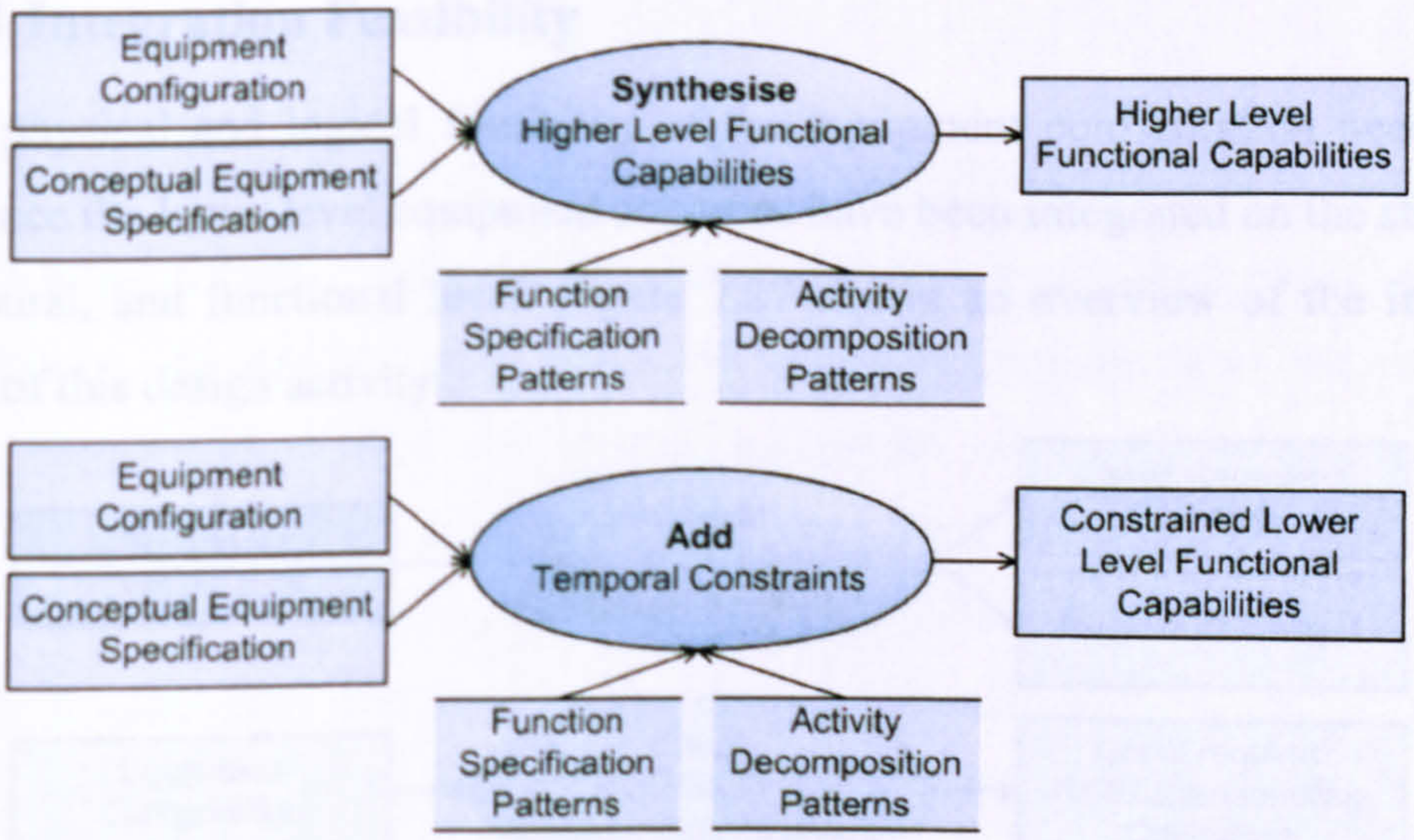


Figure 7.26 Function Synthesis Inferences

Figure 7.26 shows an overview of the function synthesis activities. The synthesis activity requires a fully physical and logical connected equipment description as input. This could be for example an assembly unit consisting of a manipulator device, a tool device, and a positioning device. It is important for the assessment of the integrated equipment that its contained lower level equipment modules match those originally defined in the conceptual equipment specification.

The synthesis of the functional capabilities is based on the *ProcessDecompositionPatterns* in conjunction with the *FunctionSpecificationPatterns* which have been defined in chapter 5 and chapter 6 respectively. The *FunctionSpecificationPatterns* relate the lower level functions back to the process model. The *ProcessDecompositionPatterns* can be used to recognise the higher level process capabilities of the translated functions. This in turn could be related back to the functional domain using the *FunctionSpecificationPatterns* on the resulting higher level activities. The *ProcessDecompositionPatterns* are also used to specify the temporal relationships between the functions based on the assembly process for which the equipment is going to be responsible.

For the assessment of whether or not the existing functional capabilities of the equipment modules actually fulfils the original requirements, it would not be necessary to derive all possible higher level functions. The approach can be more targeted since the required result is already known.

7.5.4.3 Integration Feasibility

The physical and logical feasibility of the equipment configuration needs to be tested once the lower level equipment solutions have been integrated on the structural, behavioural, and functional level. Figure 7.27 shows an overview of the input and outputs of this design activity.

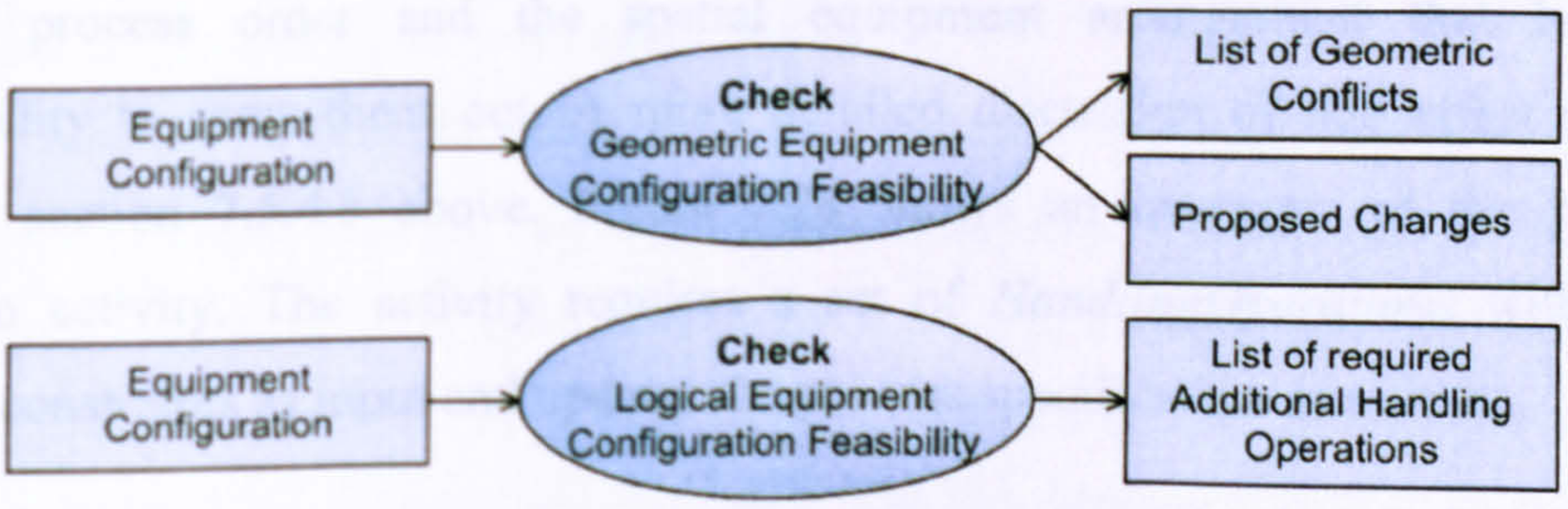


Figure 7.27 Integration Feasibility Check Inferences

The assessment of the physical feasibility ensures that there are no geometric interferences, collisions, etc. If conflicts have been found, then either different equipment solutions need to be found that do not cause a conflict, or the whole structure of the composite equipment has to be changed.

The assessment of the logical feasibility maintains the spatial temporal integrity of the configuration. This is important to ensure that the process sequence that was originally planned in the temporal space can actually be carried out in its current form by the equipment that is arranged in the spatial space. The logical feasibility assessment becomes more important on the workstation level where different equipment units need to be integrated.

Conflicts arise quite commonly from the fact that the physical location of the equipment could not have been known during the specification of the original assembly process. Consequently, some process steps might be missing that are required as the result of the configuration. These are generally handling operations that define how two successive activities can be connected in the spatial domain.

The additional operations need to be added into the original assembly process specification (see section 7.5.4.4). This would cause a re-iteration of the decomposition and integration which in turn might change the current configuration of the equipment. This iteration could take place a few times until a stable equipment-process combination can be found. The effect of this iteration is minimised however, by the use of predefined system architectures that would have taken such situations into consideration.

7.5.4.4 Adaptation of the Assembly Process Specification

This design activity addresses the need for the original assembly process specification to be adapted as a result of inconsistencies between the required temporal process order and the spatial equipment arrangement that has the responsibility to carry them out. A more detailed discussion of this effect can be found in section 7.5.4.3 above. Figure 7.28 shows an overview of the process adaptation activity. The activity requires a set of *HandlingOperations* with their temporal constraints as input and updates the process specification as a result.

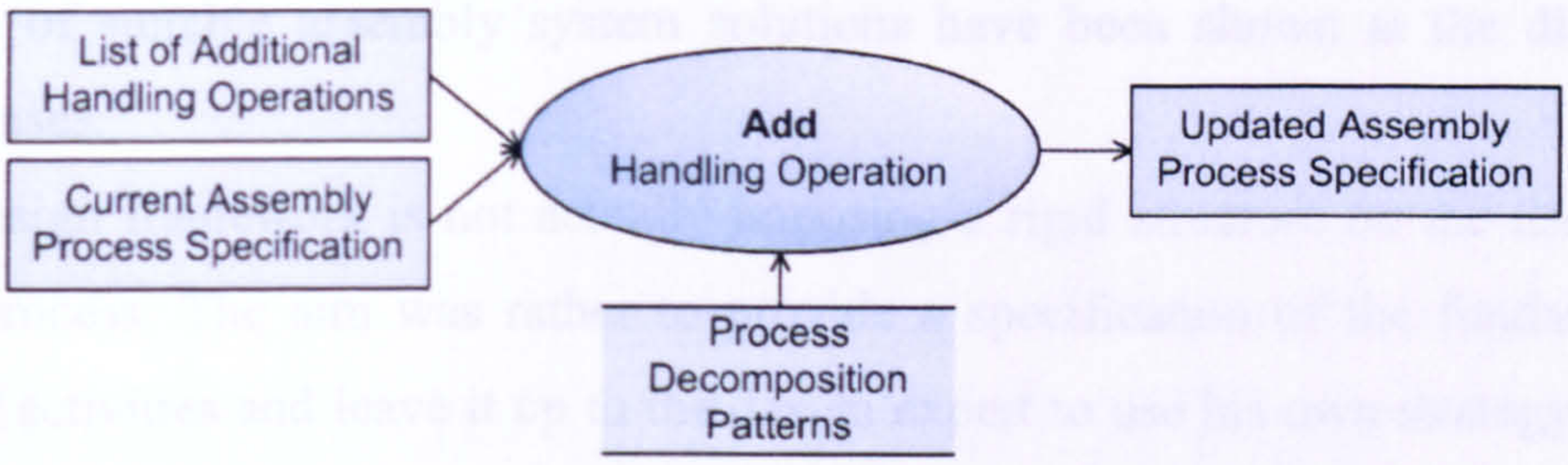


Figure 7.28 Process Specification Adaptation Inference

7.5.5 Evaluation Task

This section describes the evaluation of equipment configurations that need to take place once their technical feasibility has been determined. This task is important to make a choice between alternative solution proposals. It also gives an indication of whether it is going to be likely to find better solutions through further design iterations. This section only gives a brief overview of the required characteristics of this task since the specific details were outside the scope of this work.

The evaluation task needs to be able to address at least the cost, cycle time, and utilisation of the proposed equipment. A multi-criteria evaluation approach needs to be used to rank the proposed solutions according to their suitability.

7.6 Summary

In this chapter an integrated design approach has been specified that takes advantage of the modular assembly system domain conceptualisation proposed by the ONTOMAS framework. The main objective was to provide a design framework that allows the integrated specification of assembly processes and assembly equipment solutions. The fundamental structure for such a framework has been defined in this chapter providing detailed inference specifications for the different required design activities. The approach is based on hierarchical decomposition and synthesis which is well suited for the design of modular systems.

It has been shown how the different formalisms used in the ONTOMAS framework support the decision making during modular assembly system design. The basic aggregation, topological, taxonomy, and system theory principles are used to maintain and structure the complex knowledge accumulated during the design process. The defined relationships of the core ontology and the different design patterns help maintain the consistence of complex assembly system specification across all domains. The interaction between the specification of the assembly process and the

definition of suitable assembly system solutions have been shown at the different design phases.

The design framework is not actually imposing a rigid structure on the decision-making process. The aim was rather to provide a specification of the fundamental reasoning activities and leave it up to the design expert to use his own strategy. Only very loose precedence relationships between the decisions-making activities were outlined. They are mainly imposed by the need for specific input knowledge that is only being generated in the preceding design activity.

The design decision-making activities and general precedence constraints are well suited for the application of design optimisation, distributed decision making, and computer aided reasoning applications. The reported design framework defines only the general rules of how the ONTOMAS conceptualisation can be utilised. The specific implementation can be done using a variety of different approaches.

This chapter concludes the definition and discussion of the proposed ontology framework for the design of modular assembly systems. The next chapter is looking at the verification of the approach using an extensive verification scenario, describing industrial application of the framework, and outlining a prototype implementation of the design approach described in this chapter.

8 Illustration and Verification

8.1 Introduction

The aim of this chapter is to outline the effort that has been made towards verifying the proposed ontology framework for the design of modular assembly workstations (ONTOMAS). The ontology has been applied in several representative test cases. The results of the test cases have given an initial indication of the validity of the proposed framework.

ONTOMAS has been fully modelled in Protégé (Protégé [97]) to test the technical applicability of the framework. Protégé is an ontology definition and instantiation framework that provides a software environment for the definition of ontologies. The ontology can be instantiated and made available as a knowledge base for both internal and external utilisation. The full implementation and instantiation in the Protégé framework proved that ONTOMAS can be translated into a fully functional knowledge base structure. Protégé was also used to test some more intricate axioms and definitions that went beyond the validation of the pure technical applicability of the proposed formalisms towards the creation of a comprehensive domain theory.

ONTOMAS has also been used as the foundation for a web-based assembly system design environment which was developed for the Eureka Factory E-Race project (E-Race [32]). The prototype implementation has been used in several industrial relevant projects which indicated the appropriateness of the proposed framework.

This chapter provides further details on the verification cases starting with an elaborate demonstration example that shows the detailed instantiation of the proposed domain conceptualisation. The verification cases look at the applicability of the ontology for the design of new workstation as well as the reconfiguration or adaptation of existing workstations to new requirements. Some more details on the knowledge base development are given and the role of ONTOMAS as foundation for a web-based assembly system design framework is shown. Finally some more

dissemination and application activities are outlined which lead to the conclusions of this chapter.

8.2 Ontology Framework Verification Cases

This section provides a detailed scenario for the initial verification of the proposed ontology framework within an integrated design environment. The design of a new workstation based on product requirements and the adaptation of an existing workstation to changing requirements are demonstrated throughout this section in a step-by-step approach. This approach is felt to best illustrate the underlying formalisms defined in this work and they can be applied.

The application of ONTOMAS in representative test cases is the first step towards validating the appropriateness and relevance of the proposed ontology framework. Furthermore, it makes the framework more accessible for a wider scrutiny which is required for an in-depth validation and acceptance by domain experts. The detailed validation goes beyond the scope and time frame of this work but is expected to continue in current and planned future projects.

The diagrams used in the verification cases use the symbols defined in the symbology for illustrative purposes. Additionally some informal text definitions are used to convey the intention of the knowledge content but not its exact formalisation.

8.2.1 Design of a new assembly workstation

The design of a new assembly workstation scenario goes through both the process decomposition and the workstation configuration in a step-by-step approach. The starting point for the design of an assembly system is the definition of a set of user requirements around the product that needs to be assembled. Additionally, one of the assumptions behind this example is that the assembly process and system have already been defined using available methods. Starting from this assumption, the process is decomposed and a conceptual system structure is being defined using the ONTOMAS concepts (see chapter 4 to 6).

A product from day to day life has been chosen for the verification example to increase its comprehensiveness. The Three Pin Plug has been chosen because it illustrates quite a number of the implications that arise from the product domain concepts defined in chapter 4. Incidentally the Three Pin Plug has also been used by Rampersad [99] to demonstrate his design approach. The plug is suitable because of

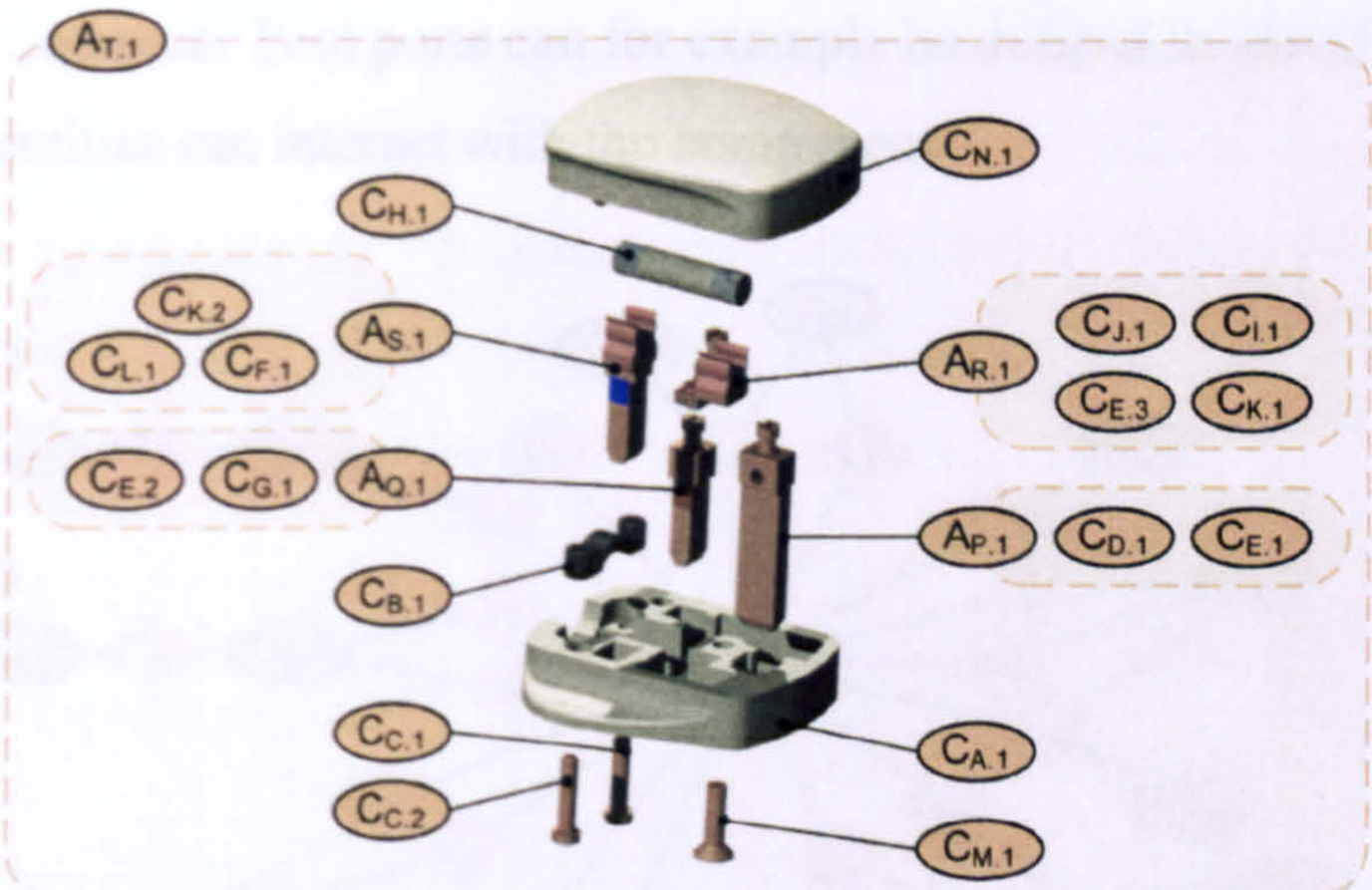
its clear and comprehensible structure. It also demonstrates most of the product domain relevant modelling requirements including hierarchical product structure (sub-assemblies), multiple occurrences of the same components within the hierarchy, and topological relationships using a number of different liaison types from all three main categories. Some aspects are not demonstrated which are more relevant for the overall assembly system layout and less for the design of assembly workstations. They include multiple occurrences of the same sub-assemblies and the definition of product families. The given set of product requirements, however, is well suited to demonstrate a sufficient range of different aspects of the assembly process and equipment specification. The plug will be used throughout this chapter to demonstrate the different aspects of the assembly workstation design process.

The test-case illustration starts with a detailed description of the product requirements. These are turned into assembly process requirements which in turn lead to the definition of equipment requirements. The process and equipment requirements definition take place in an iterative loop. The actual equipment selection and integration take place following the requirements definition. Again this is an iterative loop that can result in some more requirements changes.

8.2.1.1 Product Definition

This section shows the application of the ontology concepts that are used to define the product based user requirements for the test case. Figure 8.1 shows the hierarchical product structure of the Three Pin Plug ($A_{T.1}$) and Figure 8.2 shows the topology of the Plug. Assemblies are defined with a capital A and components with a capital C. Their suffix gives a letter for their individual id, and a number for their occurrence. The component liaisons in Figure 8.2 are defined with a capital L and a number to give them a different id. It is important to notice that the component liaisons shown in Figure 8.2 are simplified and do not show the connection ports.

While the whole Three Pin Plug is well suited to demonstrate the proposed product model, it is too complex to demonstrate the design of assembly workstations within the short space of this work. The Cable Holder subassembly ($A_{R.1}$) has been chosen to demonstrate the process and equipment related design stages. The Cable Holder has sufficient topological complexity to demonstrate the definition of different assembly process and equipment relevant aspects.



Product Structure Definition:

$A_{T.1}$: *Assembly* "Three Pin Plug" occ1
(<hasParts> ($A_{P.1}$, $A_{Q.1}$, $A_{R.1}$, $A_{S.1}$, $C_{A.1}$, $C_{B.1}$, $C_{C.1}$, $C_{C.2}$, $C_{H.1}$, $C_{M.1}$, $C_{N.1}$, L_1, L_2, L_4, L_5, L_7, L_8, L_9, L_{10}, L_{11}, L_{12}, L_{13}, L_{14}, L_{15}, L_{16}, L_{20}, L_{21}, L_{24}, L_{25}))

$A_{P.1}$: *Assembly* "Earth Pin Assembly" occ1
(<hasParts> ($C_{D.1}$, $C_{E.1}$, L_3))

$A_{Q.1}$: *Assembly* "Neutral Pin Assembly" occ1
(<hasParts> ($C_{G.1}$, $C_{E.2}$, L_6))

$A_{R.1}$: *Assembly* "Cable Holder Assembly" occ1
(<hasParts> ($C_{E.3}$, $C_{I.1}$, $C_{J.1}$, $C_{K.1}$, L_{17}, L_{18}, L_{19}))

$A_{S.1}$: *Assembly* "Life Pin Assembly" occ1
(<hasParts> ($C_{F.1}$, $C_{L.1}$, $C_{K.2}$, L_{22}, L_{23}))

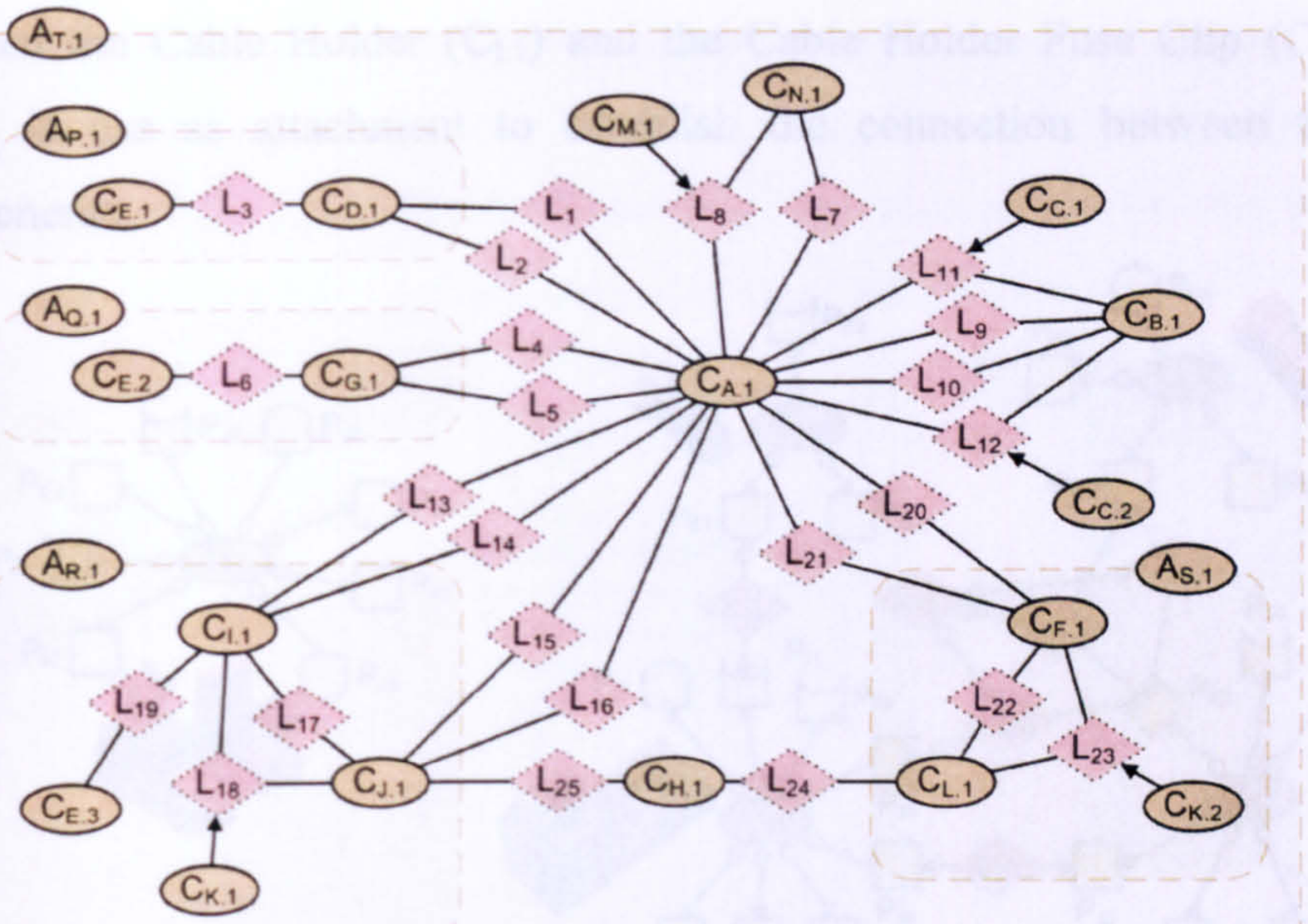
Product Component Definition:

$C_{A.1}$: *Component* "Base" occ1
$C_{B.1}$: *Component* "Cord Grip" occ1
$C_{C.1}$: *Component* "Cord Grip Screw" occ1
$C_{C.2}$: *Component* "Cord Grip Screw" occ2
$C_{D.1}$: *Component* "Earth Pin" occ1
$C_{E.1}$: *Component* "Fixing Screw" occ1
$C_{E.2}$: *Component* "Fixing Screw" occ2
$C_{E.3}$: *Component* "Fixing Screw" occ3
$C_{F.1}$: *Component* "Life Pin" occ1
$C_{G.1}$: *Component* "Neutral Pin" occ1
$C_{H.1}$: *Component* "Fuse" occ1
$C_{I.1}$: *Component* "Cable Holder" occ1
$C_{J.1}$: *Component* "Cable Holder Fuse Clip" occ1
$C_{K.1}$: *Component* "Rivet" occ1
$C_{K.2}$: *Component* "Rivet" occ2
$C_{L.1}$: *Component* "Pin Fuse Clip" occ1
$C_{M.1}$: *Component* "Cover Screw" occ1
$C_{N.1}$: *Component* "Cover" occ1

Figure 8.1 Product Structure of a Three Pin Plug

Figure 8.3 shows the more detailed definition of the Cable Holder Assembly ($A_{R.1}$). The liaison concept defines the relationships only between the predefined ports of the components. This additional detail has been left out in Figure 8.2 to reduce the complexity of the diagram. The ports make certain features of the components available to be connected to from the outside world. This connection could be to another component or to some kind of equipment like a tool for example. For instance the Cable Holder ($C_{I.1}$) has a threaded hole that provides the connection point (P_{11}) for screws of a specific size. Other features of the Cable Holder such as holes or combinations of surfaces are also defined as ports since they are also intended to provide connection points to other components. It becomes clear at this point that the definition of ports is very much based on the intention of the component designer. Ports allow the product and system designer to restrict the interaction of a component

with its environment. Tool ports can for example be defined to specify where and how equipment entities can interact with the component.



Conceptual Product Liaison Definition:

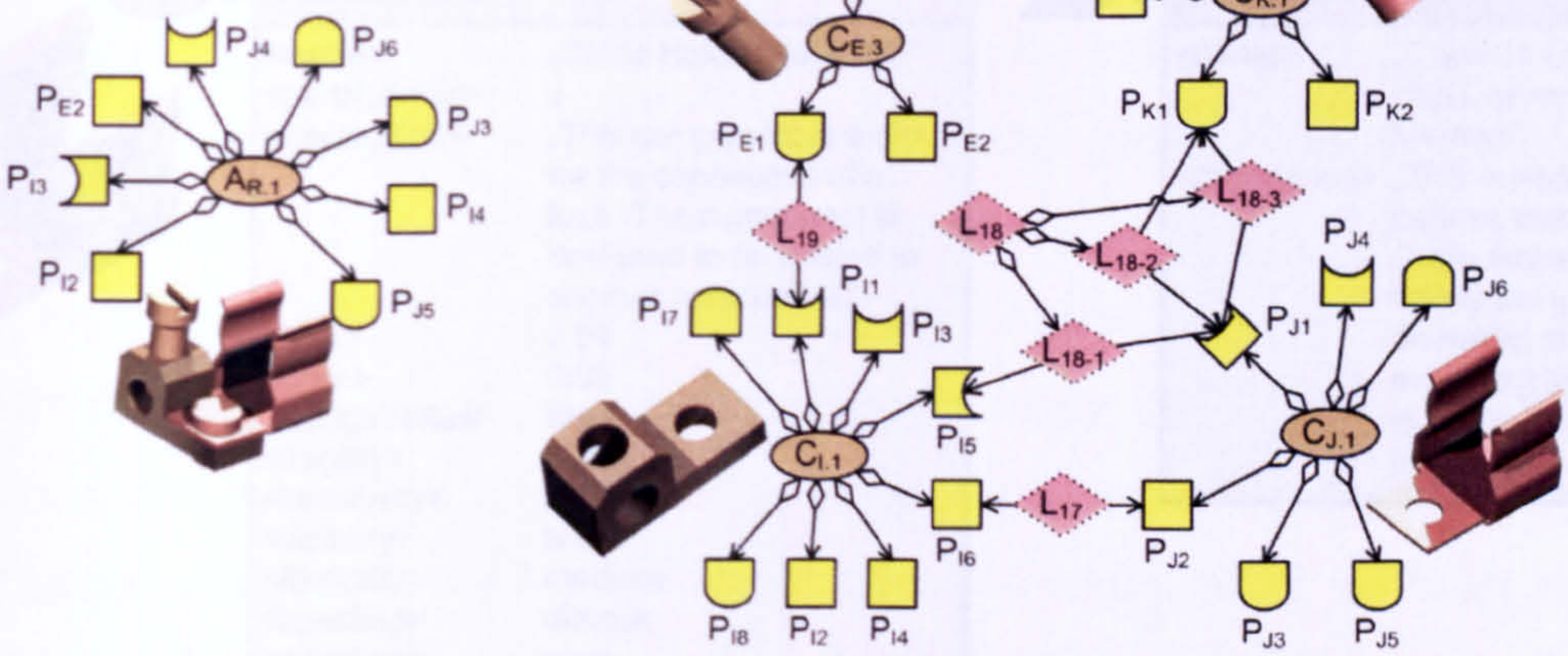
L_1: *LooseFitLiaison* (<connects> ($C_{A.1}$, $C_{D.1}$))
L_2: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{D.1}$))
L_3: *ScrewFitLiaison* (<connects> ($C_{D.1}$, $C_{E.2}$))
L_4: *LooseFitLiaison* (<connects> ($C_{A.1}$, $C_{G.1}$))
L_5: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{G.1}$))
L_6: *ScrewFitLiaison* (<connects> ($C_{G.1}$, $C_{E.1}$))
L_7: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{N.1}$))
L_8: *ScrewConnectionLiaison*
(<connects> ($C_{A.1}$, $C_{N.1}$, $C_{M.1}$), <attachments> ($C_{M.1}$))
L_9: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{B.1}$))
L_{10}: *LooseFitLiaison* (<connects> ($C_{A.1}$, $C_{B.1}$))
L_{11}: *ScrewConnectionLiaison*
(<connects> ($C_{A.1}$, $C_{B.1}$, $C_{C.1}$), <attachments> ($C_{C.1}$))
L_{12}: *ScrewConnectionLiaison*
(<connects> ($C_{A.1}$, $C_{B.1}$, $C_{C.2}$), <attachments> ($C_{C.2}$))
L_{13}: *LooseFitLiaison* (<connects> ($C_{A.1}$, $C_{I.1}$))
L_{14}: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{I.1}$))
L_{15}: *LooseFitLiaison* (<connects> ($C_{A.1}$, $C_{J.1}$))
L_{16}: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{J.1}$))
L_{17}: *ContactLiaison* (<connects> ($C_{I.1}$, $C_{J.1}$))
L_{18}: *RivetLiaison*
(<connects> ($C_{I.1}$, $C_{J.1}$, $C_{K.1}$), <attachments> ($C_{K.1}$))
L_{19}: *ScrewFitLiaison* (<connects> ($C_{I.1}$, $C_{E.3}$))
L_{20}: *LooseFitLiaison* (<connects> ($C_{A.1}$, $C_{F.1}$))
L_{21}: *ContactLiaison* (<connects> ($C_{A.1}$, $C_{F.1}$))
L_{22}: *ContactLiaison* (<connects> ($C_{F.1}$, $C_{L.1}$))
L_{23}: *RivetConnectionLiaison*
(<connects> ($C_{F.1}$, $C_{L.1}$, $C_{K.2}$), <attachments> ($C_{K.2}$))
L_{24}: *SnapFitLiaison* (<connects> ($C_{L.1}$, $C_{H.1}$))
L_{25}: *SnapFitLiaison* (<connects> ($C_{J.1}$, $C_{H.1}$))

Figure 8.2 Product Topology of a Three Pin Plug

Note that not all the ports of the components in the Cable Holder subassembly ($A_{R.1}$) are being used inside the assembly. Some of them are exposed to the outside of the assembly to enable it to be connected to the other subassemblies and components in the overall Plug assembly. For example the Cable Holder Fuse Clip ($C_{J.1}$) connects to the Fuse ($C_{H.1}$) and therefore exposes the clip outside the assembly definition (P_{J4}).

Three different types of liaisons connect the components in the Cable Holder subassembly. Liaison L_{17} defines that there is contact between the Cable Holder ($C_{I.1}$) and the Cable Holder Fuse Clip ($C_{J.1}$). This liaison is required to indicate that these two components have two touching surfaces. Liaison L_{19} is a screw fit liaison. The reason for choosing this type and not the screw connection liaison type is that the

Fixing Screw ($C_{E.3}$) is not being used to connect two other components. $C_{E.3}$ is therefore not used in the role of an attachment. L_{18} defines the rivet connection between the Cable Holder ($C_{I.1}$) and the Cable Holder Fuse Clip ($C_{J.1}$). The rivet ($C_{K.1}$) is use as attachment to establish the connection between the two other components.



Conceptual Product Definition:

$A_{R.1}$: *Assembly* (<hasPorts> (P_{E2}, P_{I2}, P_{I3}, P_{I4}, P_{J3}, P_{J4}))
$C_{E.3}$: *Component* (<hasPorts> (P_{E1}, P_{E2}))
$C_{I.1}$: *Component* (<hasPorts> (P_{I1}, P_{I2}, P_{I3}, P_{I4}, P_{I5}, P_{I6}))
$C_{J.1}$: *Component* (<hasPorts> (P_{J1}, P_{J2}, P_{J3}))
$C_{K.1}$: *Component* (<hasPorts> (P_{K1}, P_{K2}, P_{K3}, P_{K4}))
L_{17}: *ContactLiaison* (<connects> (P_{I6}, P_{J2}))
L_{18}: *RivetLiaison*
(<connects> (P_{I5}, P_{J1}, P_{K1}), <attachments> ($C_{K.1}$),
<hasParts> ($L_{18\text{-}1}$, $L_{18\text{-}2}$, $L_{18\text{-}3}$))
$L_{18\text{-}1}$: *LooseFitLiaison* (<connects> (P_{I5}, P_{J1}))
$L_{18\text{-}2}$: *ContactLiaison* (<connects> (P_{J1}, P_{K1}))
$L_{18\text{-}3}$: *LooseFitLiaison* (<connects> (P_{J1}, P_{K1}))
L_{19}: *ScrewFitLiaison* (<connects> (P_{I6}, P_{E1}))

Port Definitions:

P_{E1}: *ComponentPort* (<direction> male)
P_{E2}: *ComponentPort* (<direction> neutral)
P_{E3}: *ToolPort* (<direction> female)
P_{I1}: *ComponentPort* (<direction> female)
P_{I2}: *ComponentPort* (<direction> neutral)
P_{I3}: *ComponentPort* (<direction> female)
P_{I4}: *ComponentPort* (<direction> neutral)
P_{I5}: *ComponentPort* (<direction> female)
P_{I6}: *ComponentPort* (<direction> neutral)
P_{I7}: *ToolPort* (<direction> male)
P_{I8}: *ToolPort* (<direction> male)
P_{J1}: *ComponentPort* (<direction> female)
P_{J2}: *ComponentPort* (<direction> neutral)
P_{J3}: *ComponentPort* (<direction> male)
P_{J4}: *ComponentPort* (<direction> female)
P_{J5}: *ToolPort* (<direction> male)
P_{J6}: *ToolPort* (<direction> male)
P_{K1}: *ComponentPort* (<direction> male)
P_{K2}: *ToolPort* (<direction> neutral)
P_{K3}: *ToolPort* (<direction> neutral)
P_{K4}: *ToolPort* (<direction> male)

Figure 8.3 Topology definition of the Cable Holder Subassembly

The rivet connection establishes a permanent link between the other two components. However, before it does, the other parts need to be brought in a specific relationship to each other. This relationship is defined by L_{17}. Additionally the rivet needs to be brought into the right position before the riveting process can take place. The rivet connection has been broken down into a more detailed description to define this aspect. The sub-liaisons $L_{18\text{-}1}$ to $L_{18\text{-}3}$ define the geometric relationship between the Rivet C_{K1} and the two components it is going to connect ($C_{I.1}$ and $C_{J.1}$).

Each component in the assembly is specified only to the necessary level of detail required for the configuration of modular assembly workstations (see also discussion in section 4.3). Figure 8.4 shows the high level specification of the Cable Holder Fuse Clip (C_J) including its delivery method ($T_{D.J}$). The definition is the same for all occurrences of the Cable Holder Fuse Clip. The assumption for this work is that all the components are fully defined including their delivery method.

$C_{J.1}$

Component C_J

<name>	„Cable Holder Fuse Clip“
<partNumber>	J
<description>	„This component is a clip for the connection of a fuse. The component is designed to be riveted to another component.“
<cost>	0.04
<price>	0.05
<designStatus>	fixed
<fragility>	high
<sensitivity>	medium
<visibility>	small
<flexibility>	medium
<handling>	difficult
<orienting>	easy
<deliveryMethod>	($T_{D.J}$)

$T_{D.J}$

DeliveryTask $T_{D.J}$

<name>	„Cable Holder Fuse Clip Delivery Method“
<description>	„This method defines that the Cable Holder Fuse Clip is being delivered to the assembly process in boxes of 200 pieces.“

Figure 8.4 High level component attribute definition example

The assembly specific characteristics of the components are defined in qualitative terms based on the experience of the production engineer. The Cable Holder Fuse Clip for example is made from copper and is quite thin and intricate. The fragility of it is therefore judged to be *'high'*, the sensitivity which is mainly referring to how easy the surface can be damaged as *'medium'* since copper is relatively soft, and the flexibility as *'medium'* due to the material properties of copper. These characteristics could be inferred from a more detailed definition of the components' geometry and material properties. This however is outside the scope of this work. The visibility of C_J is defined as *'small'* since the component is inside the plug and can only be seen when the plug is opened and not during the normal use of the plug. Handle-ability and orient-ability are judged by the assembly system designer. The Cable Holder Fuse Clip is relatively *'easy'* to orient since it has easily definable stable positions. Its handle-ability on the contrary is *'difficult'* due to its complex geometry and high fragility.

The Cable Holder Fuse Clip is assumed to come from an outside manufacturing process and is being delivered in boxes of 200 pieces. The <deliveryMethod> attribute of the component is linked to a *DeliveryTask* ($T_{D.J}$) that specifies all the

relevant details of how the component arrives at the assembly process. This information is fundamental for the selection of the right feeding equipment.

8.2.1.2 Initial Assembly Process and Workstation Definition

The focus of the validation scenario is the sub-task level definition of assembly processes and the corresponding sub-workstation level configuration of modular assembly equipment. Hence it has been assumed that the higher level assembly process order and the wider assembly system structure have already been defined.

A more detailed description of the task level assembly processes requirements and workstation scopes for the Cable Holder Assembly ($A_{R.1}$) are given in Figure 8.5. The overall requirements of the whole Cable Holder Assembly assembly process are abstractly defined by *Task* $T_{A.R}$. The *Task* has as many inputs as the number of components that need to be assembled. The inputs are linked to the *DeliveryTasks* of those components. They have to match with the <deliveryMethod> attributes of the components. The *Task* has only one output, the finished assembly. In this particular case the output of $T_{A.R}$ is linked to a follow up assembly process that is linked with a *TransportationTask* ($T_{T.R}$).

The establishment of each liaison in $A_{R.1}$ is defined through a separate *AssemblyTask*. They are linked to the liaisons with their <requirements> attribute. It is worth noting that the two-state *RivetingLiaison* (L_{18}) needs two *AssemblyTask* (T_{A2} and T_{A3}) to be completed; one specifies the requirements for establishing the first state and the second to define the transition to the second state.

The Cable Holder Assembly ($A_{R.1}$) can be put together in two different ways. Those two alternatives task sequences are defined by the *ProcessVariants* V_{S1} and V_{S2}. The *ProcessVariants* only defined OR-branches in the process hierarchy. Consequently they do not have input and output ports themselves but rather use the ones of their parent node.

In this particular example the *AssemblyTask* in both sequences are the same. They are only arranged in a different temporal order. This, however, is not always the case. An *AssemblyTask* is defined by its liaison requirements and it is possible the different *ProcessVariants* will require different sets of liaisons to be established. In such cases they would contain different *AssemblyTask* definitions.

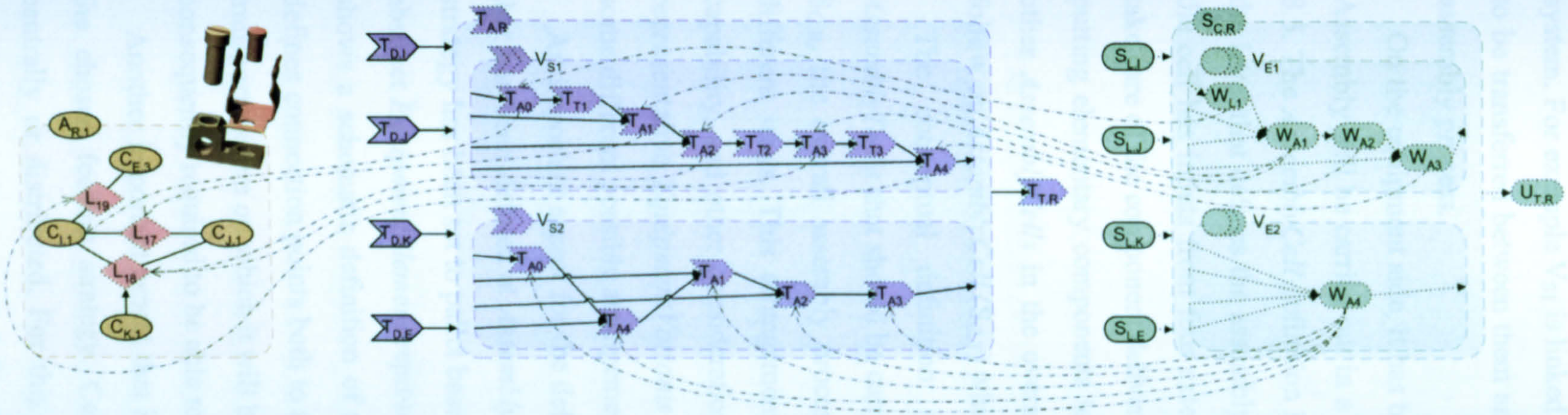


Figure 8.5 Illustration of the Initial Assembly Process and Workstation Requirements

Conceptual Assembly Process Definition:

$T_{A.R}$: *AssemblyTask*: Assemble Fuse Connector Assembly
(<requirements> ($A_{R.1}$), <hasParts> (V_{S1}, V_{S2}))

V_{S1}: *ProcessVariant*
(<hasParts> (T_{A0}, T_{T1}, T_{A1}, T_{T2}, T_{A2}, T_{A3}, T_{T3}, T_{A4},
TR_0, TR_1, TR_2, TR_3, TR_4, TR_5, TR_6, TR_7, TR_8, TR_9, TR_{10}, TR_{11}))

V_{S2}: *ProcessVariant*
(<hasParts> (T_{A0}, T_{A2}, T_{A3}, T_{A1}, TR_0, ...))

T_{A0}: *LoadingTask* (<requirements> (C_I))

T_{A1}: *AssemblyTask* (<requirements> (L_{17}), <basePart> ($C_{I.1}$))

T_{A2}: *AssemblyTask* (<requirements> (L_{18}), <basePart> ($C_{I.1}$))

T_{A3}: *AssemblyTask* (<requirements> (L_{18}), <basePart> ($C_{I.1}$))

T_{A4}: *AssemblyTask* (<requirements> (L_{19}), <basePart> ($C_{I.1}$))

T_{T1}: *TransportTask* (<requirements> ($C_{I.1}$))

T_{T2}: *TransportTask* (<requirements> (A {$C_{I.1}$, $C_{J.1}$, $C_{K.1}$, L_{17}, $L_{18.1}$}))

T_{T3}: *TransportTask* (<requirements> (A {$C_{I.1}$, $C_{J.1}$, $C_{K.1}$, L_{17}, $L_{18.2}$}))

TR_0: *TemporalRelation* (T_{A0}.input1, $T_{A.R}$.input1, after)

TR_1: *TemporalRelation* (T_{T1}.input1, T_{A0}.output1, after)

TR_2: *TemporalRelation* (T_{A1}.input1, T_{T1}.output1, after)

TR_3: *TemporalRelation* (T_{A1}.input2, $T_{A.R}$.input2, after)

TR_4: *TemporalRelation* (T_{A2}.input1, T_{A1}.output1, after)

TR_5: *TemporalRelation* (T_{A2}.input2, $T_{A.R}$.input3, after)

TR_6: *TemporalRelation* (T_{T2}.input1, T_{A2}.output1, after)

TR_7: *TemporalRelation* (T_{A3}.input1, T_{T2}.output1, after)

TR_8: *TemporalRelation* (T_{T3}.input1, T_{A3}.output1, after)

TR_9: *TemporalRelation* (T_{A4}.input1, T_{T3}.output1, after)

TR_{10}: *TemporalRelation* (T_{A4}.input2, $T_{A.R}$.input4, after)

TR_{11}: *TemporalRelation* (T_{A4}.output1, $T_{A.R}$.output1, before)

...

Conceptual Assembly Equipment Definition:

$S_{C.R}$: *AssemblyCell*: Assemble Fuse Connector Assembly
(<requirements> ($T_{A.R}$), <hasParts> (V_{E1}, V_{E2}))

V_{E1}: *EquipmentVariant*
(<requirements> (V_{S1}),
<hasParts> (W_{L1}, W_{A1}, W_{A2}, W_{A3}, MF_1, MF_2, MF_3, MF_4, MF_6, MF_7, MF_8))

V_{E2}: *EquipmentVariant*
(<requirements> (V_{S2}), <hasParts> (W_{A3}, MF_9, ...))

W_{L1}: *LoadingStation* (<requirements> (T_{A0}))

W_{A1}: *AssemblyWorkstation* (<requirements> (T_{A1}, T_{A2}))

W_{A2}: *AssemblyWorkstation* (<requirements> (T_{A3}))

W_{A3}: *AssemblyWorkstation* (<requirements> (T_{A4}))

W_{A4}: *AssemblyWorkstation* (<requirements> (T_{A0}, T_{A4}, T_{A1}, T_{A2}, T_{A3}))

MF_1: *MaterialFlow* ($S_{C.R}$.port1, W_{L1}.port1)

MF_2: *MaterialFlow* (W_{L1}.port2, W_{A1}.port1)

MF_3: *MaterialFlow* ($S_{C.R}$.port2, W_{A1}.port2)

MF_4: *MaterialFlow* ($S_{C.R}$.port3, W_{A1}.port3)

MF_5: *MaterialFlow* (W_{A1}.port4, W_{A2}.port1)

MF_6: *MaterialFlow* (W_{A2}.port2, W_{A3}.port1)

MF_7: *MaterialFlow* ($S_{C.R}$.port4, W_{A3}.port2)

MF_8: *MaterialFlow* (W_{A3}.port3, $S_{C.R}$.port5)

MF_9: *MaterialFlow* ($S_{C.R}$.port1, W_{A4}.port1)

...

Each *ProcessVariant* can also contain additional *LogisticTasks*. The requirement for *LogisticTasks* normally arises from the physical arrangement of the assembly

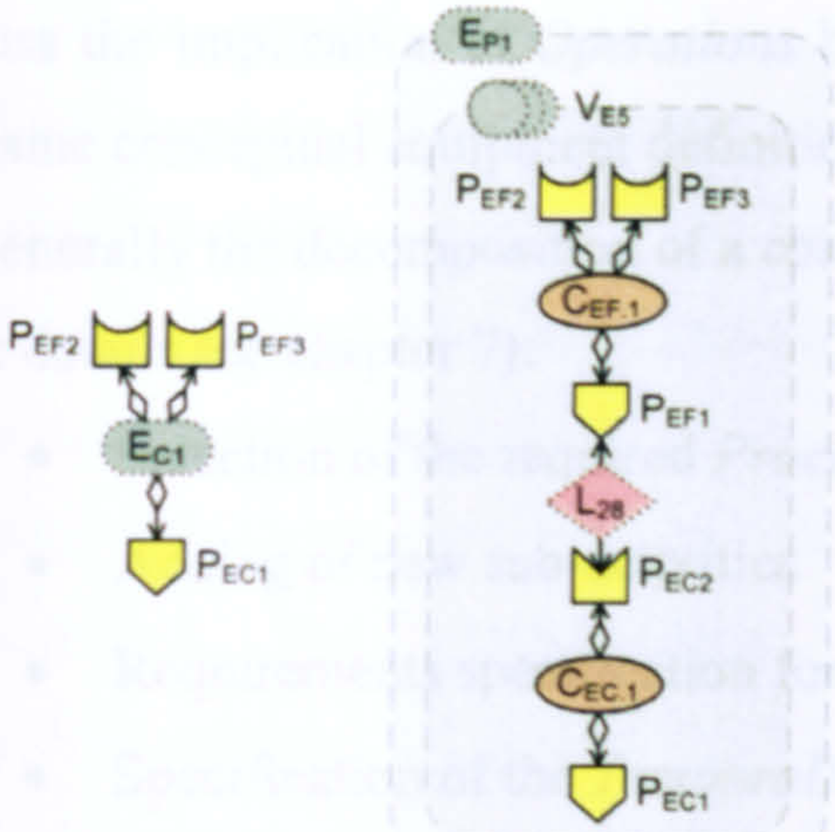
system. For example V_{S1} is linked to several *Workstations*. This requires the assembly to be transferred between them and consequently adds *TransportTasks* to the overall assembly process.

On the equipment side, it has been assumed that the assembly of the Cable Holder Assembly will be carried out in a separate *AssemblyCell* ($S_{C.R}$) as indicated in Figure 8.5. The *AssemblyCell* definition has to reflect the assembly process requirements on the level that defines the assembly of the Cable Holder Assembly ($T_{A.R}$). As a result, the cell has inputs from four external logistic systems ($S_{L.I}$, $S_{L.J}$, $S_{L.K}$, and $S_{L.E}$) that take care of the component delivery. This is the case because the *AssemblyCell* is only putting elementary components together. Otherwise, some inputs would come from other *AssemblyCells* in the overall *AssemblySystem*. The output of the cell is to a follow up *AssemblyCell* ($S_{C.T}$) which is connected via a *TransportUnit* ($U_{T.R}$).

The conceptual definition of an *AssemblyWorkstation* defines a set of *AssemblyTasks* that should be carried out at the same physical location. In the case of $S_{C.R}$, the overall assembly process could be allocated to *AssemblyWorkstations* in different ways. This arrangement is normally subject to line-balancing, technical capability, and other considerations which are outside the scope of this work. Two representative *EquipmentVariants* (V_{E1} and V_{E2}) are depicted in Figure 8.5 to show some different possible arrangements without considering those constraints.

An important aspect for the definition of the *Workstations* is the transport strategy that has been defined at *AssemblyCell* level. In this particular example the transport strategy has been set to pallet based transportation. This results in the definition of an abstract *EquipmentElement* representing the pallet type work piece carrier. Figure 8.6 shows a schematic definition of the pallet (E_{P1}) for this *AssemblyCell*. It basically defines connection points both to the *Assembly* ($A_{R.1}$) that will be placed on it and the transport system on which it will be moved. Every *Workstation* in the *AssemblyCell* is consequently required to be able to handle this kind of work piece carriers.

Another important aspect that influences the design of an *AssemblyWorkstation* is the chosen feeding strategy. Components could for example either be supplied centrally or distributed. For this particular example it has been assumed that the components will be supplied directly at each *Workstation*.



Conceptual Equipment Definitions:
E_{P1}: *WorkpieceCarrier*
(<hasParts> (V_{E5}), <hasPorts> (P_{EC1}, P_{EF2}, P_{EF3}))
V_{E5}: *EquipmentVariant*
(<hasParts> ($C_{EF.1}$, $C_{EC.1}$, L_{28}))
$C_{EC.1}$: *Component* "Pallet Base" occ1
$C_{EF.1}$: *Component* "Component Fixture" occ1
P_{EC1}: *EquipmentPort* (<direction> male)
P_{EC2}: *EquipmentPort* (<direction> female)
P_{EF1}: *EquipmentPort* (<direction> male)
P_{EF2}: *ToolPort* (<direction> female)
P_{EF3}: *ToolPort* (<direction> female)
L_{28}: *Liaison* (<connects> (P_{EC2}, P_{EF1}))

Figure 8.6 Conceptual Work Piece Carrier Definition

The *EquipmentVariant* V_{E1} splits the assembly of the Cable Holder Assembly $A_{R.1}$ into three *AssemblyWorkstations* (W_{A1}, W_{A2}, and W_{A3}) and a *LoadingStation* (W_{L1}). W_{A1} has been assigned the responsibility for two *AssemblyTasks* (T_{A1} and T_{A2}) because their intermediate state is unstable. The *LoadingTask* defines the requirements for placing the first part on the work piece carrier. The *EquipmentVariant* V_{E2} shows the case in which all four *AssemblyTasks* are carried out by the same *AssemblyWorkstation*. It can be seen from the diagram that there is a distinct relationship between the *EquipmentVariants* and the *ProcessVariants*. Each *EquipmentVariant* requires its own *ProcessVariant* because the *ProcessVariant* needs to incorporate specific additional requirements arising from the conceptual system specification.

8.2.1.3 Assembly Task Decomposition

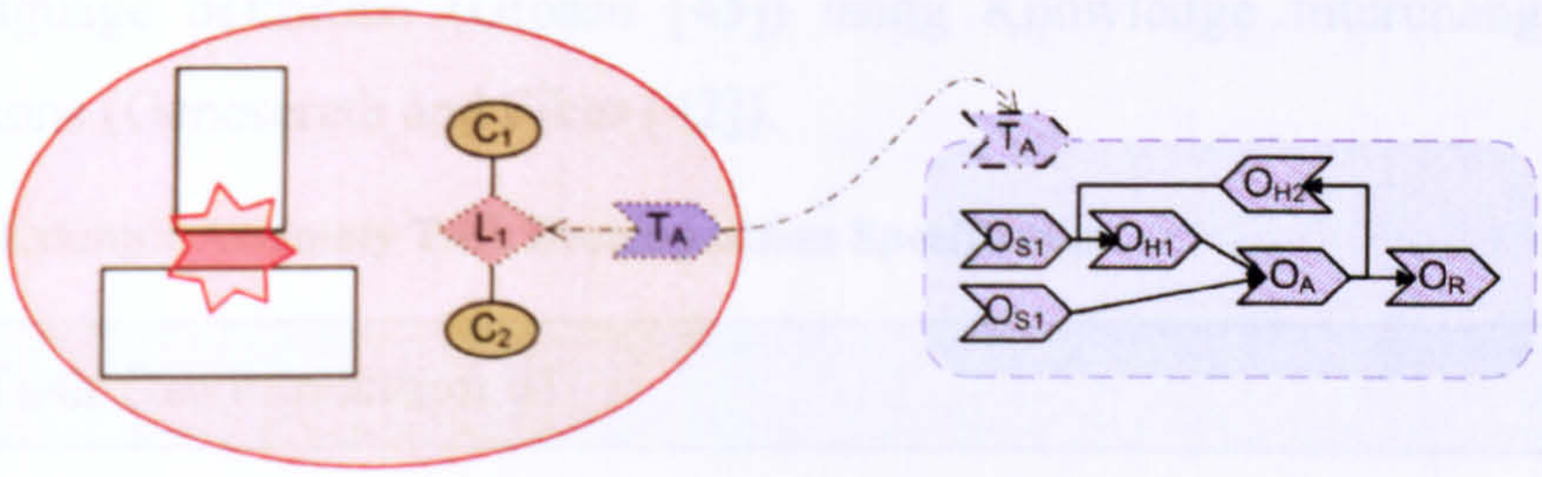
The first step during the specification of the assembly process at sub-task level is the decomposition of the *AssemblyTasks* into *Operations*. The condition for the decomposition to take place is that the higher level requirements of the *Task* have been defined to the extent discussed in the previous section.

The insertion of the Rivet $C_{K.1}$ defined by the *AssemblyTask* (T_{A2}) has been chosen to illustrate the decomposition because this *AssemblyTask* has a sufficient level of complexity to show the wider application of the proposed ontology framework. The *AssemblyTask* (T_{A2}) defines the requirements for more than one geometric relationship to be established between the Rivet $C_{K.1}$ and the other two components ($C_{I.1}$ and $C_{J.1}$) already on the pallet (E_{P1}). Furthermore, T_{A2} has been allocated together with T_{A1} to the same *AssemblyWorkstation*. This opens the scope to illustrate and

discuss the implication of *Operations* belonging to different *Tasks* to be assigned to the same conceptual equipment definitions (for more details see section 8.2.1.5).

Generally the decomposition of a complex *Activity* requires the following steps (for more details see chapter 7):

- Selection of the required *ProcessDecompositionPattern*
- Adding of new sub-activities
- Requirements specification for the new sub-activities
- Specification of the *TemporalRelations* between the sub-activities



Decomposition Pattern:
T_A: *ProcessDecompsitionPattern*
(<individualType> *AssemblyTask*,
<activityConstraints> (O_{S1}, O_{S2}, O_A, O_R, O_{H1}, O_{H2}),
<temporalConstraints> (TR_1, TR_2, TR_3, TR_4, TR_5, TR_6, TR_7))
O_{S1}: *ActivityConstraint* (<individualType> *SupplyOperation*)
O_{S2}: *ActivityConstraint* (<individualType> *SupplyOperation*)
O_A: *ActivityConstraint* (<individualType> *AssemblyOperation*)
O_R: *ActivityConstraint* (<individualType> *RemoveOperation*)
O_{H1}: *ActivityConstraint* (<individualType> *HandlingOperation*)
O_{H2} *ActivityConstraint* (<individualType> *HandlingOperation*)
TR_1: *TemporalConstraint* (O_A, O_{S1}, before)
TR_2: *TemporalConstraint* (O_A, O_{S2}, before)
TR_3: *TemporalConstraint* (O_A, O_R, after)
TR_4: *TemporalConstraint* (O_{H1}, O_{S1}, before)
TR_5: *TemporalConstraint* (O_{H1}, O_A, after)
TR_6: *TemporalConstraint* (O_{H1}, O_{H2}, before)
TR_7: *TemporalConstraint* (O_{H2}, O_A, before)

Figure 8.7 Schematic Definition of an Assembly Task

The *ProcessDecompositionPattern* is used to maintain the consistency of the sub-activities and their temporal and parametric relationships. The task level is still quite abstract in terms of the activities that the needed equipment is required to do. Consequently, there are not too many decomposition alternatives at this level. Generally each assembly task can be broken down into *AssemblyOperation*(s), *SupplyOperations*, and a *RemovalOperation*. The number of *AssemblyOperations* depends on the number of *Liaisons* that need to be established and the number of *SupplyOperations* on the number of components or sub-assemblies that are being put together. The task decomposition can be extended by *HandlingOperations* depending on the spatial arrangement of the equipment entities that will be responsible for the other operations listed above. For example, if a component is not supplied at the point

where it is needed for the assembly to take place, then an additional *HandlingOperation* will be required to overcome the spatial difference. The need for *HandlingOperations*, however, only becomes apparent during later iterations when actual equipment modules have been allocated the responsibilities for the different operations (see section 8.2.1.10).

Figure 8.7 shows the principle definition of a *ProcessDecompSpec* for an *AssemblyTask*. The corresponding formal definition of the *AssemblyTask ProcessDecompSpec* is listed in Table 8.1. The definition is based on the Protégé Axiom Language definition (Grosso [45]) using Knowledge Interchange Format (KIF) notations (Genesereth and Fikes [42]).

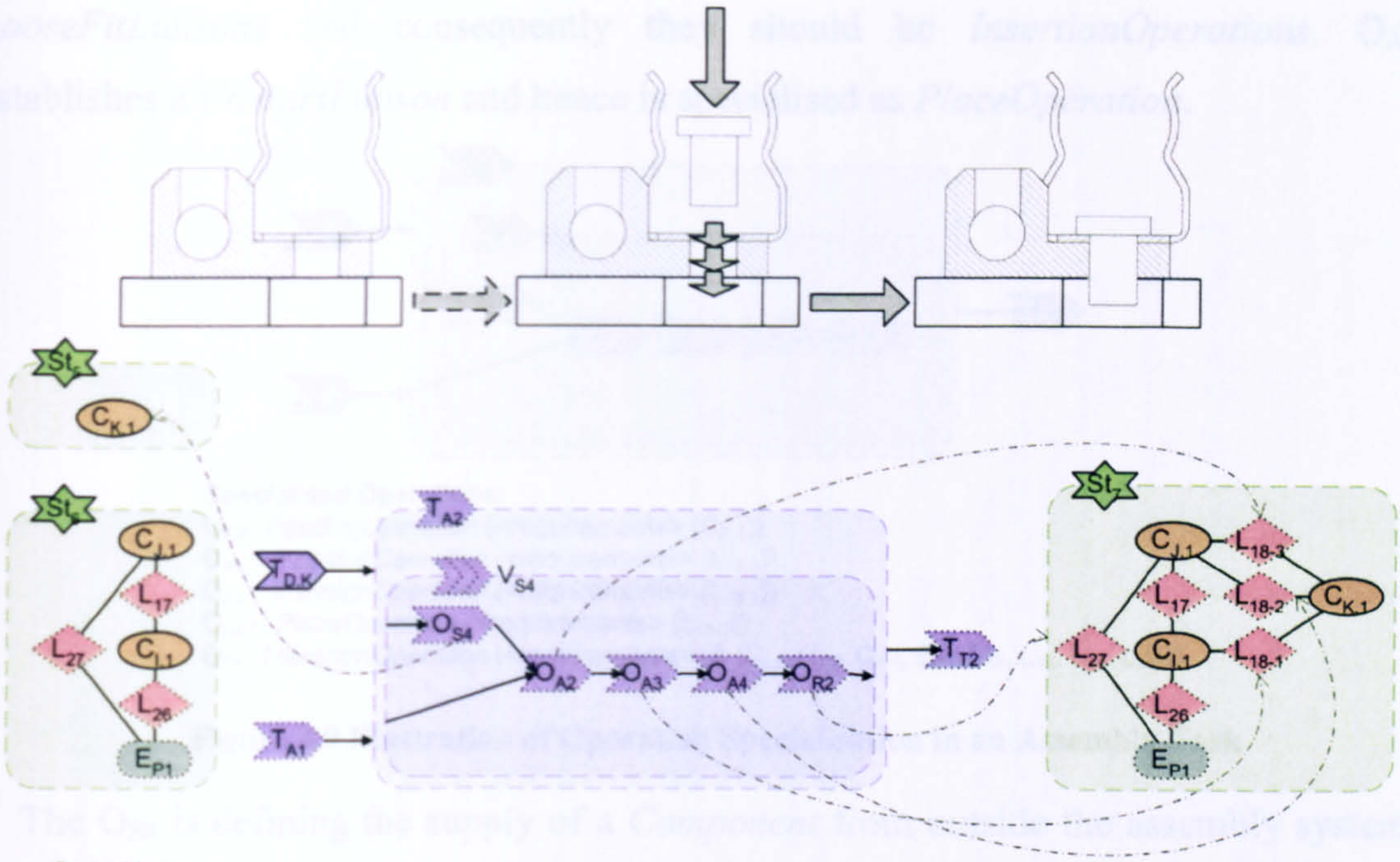
Table 8.1 Example Assembly Task Decomposition Specification

Assembly Task Decomposition 01	
Description	Every *AssemblyTask* needs to have at least one *AssemblyOp* and a *SupplyOp* and *RemoveOp* for each of the *Components* or *Assemblies* that it is connecting.
Individual Type	**AssemblyTask**
Range	**(defrange ?assyTask :FRAME AssemblyTask)** (defrange ?L :FRAME Liaison) (defrange ?C :FRAME Product) (defrange ?assyOp :FRAME AssemblyOp) (defrange ?supplyOp :FRAME SupplyOp) (defrange ?removeOp :FRAME RemoveOp) (defrange ?tempRel1 :FRAME TemporaryRelationship) (defrange ?tempRel2 :FRAME TemporaryRelationship)
Statement	**(forall ?assyTask** (forall ?L (forall ?C (=> (and (requirements ?assyTask ?L) (connects ?L ?C) (or (instance-of ?C Component) (instance-of ?C Assembly)))) (and (exists ?assyOp (and (requirements ?assyOp ?L) (hasParts ?assyTask ?assyOp))) (exists ?supplyOp (requirements ?supplyOp ?C)) (exists ?removeOp (requirements ?removeOp ?C)) (exists ?tempRel1 (and (type ?tempRel1 "before") (from ?tempRel1 ?supplyOp)

```
                    (to ?tempRel1 ?assyOp)
                    (hasParts ?assyTask ?tempRel1)
                  ))
                  (exists ?tempRel2 (and
                    (type ?tempRel2 "after")
                    (from ?tempRel2 ?removeOp)
                    (to ?tempRel2 ?assyOp)
                    (hasParts ?assyTask ?tempRel2)
                  ))
                )
              ))
            )
```

Ultimately the pattern has to reflect the constraints arising from the actually available equipment capabilities in the chosen system architecture. The temporal order on the process side is loosely related to the physical structure on the equipment side. For example if the chosen architecture only allows the use of free-flow-conveyor-based workstations with a straight flow through the station then this aspect needs to be reflected already during the process decomposition to avoid unnecessary iterations. *TransportOperations* that correspond to the conveyor characteristics would have to be required by the *ProcessDecompositionPattern*. The mechanism of defining the *ProcessDecompositionPatterns* is outside the scope of this work and it has been assumed that the architectural constraints have been taken into consideration during the definition of the patterns.

The chosen exemplary *AssemblyTask* T_{A2} has been decomposed according to these guidelines. Figure 8.8 shows the *Operations* of T_{A2}. In this specific case, the *AssemblyTask* requires three *AssemblyOperations* (O_{A2} to O_{A4}) that define the establishment of each liaison. The task has only one extra *SupplyOperation* (O_{S4}) because the preceding *AssemblyTask* has already been allocated to the same workstation. Consequently, one of the input components has already been supplied in the previous task. Finally, there the T_{A2} contains one *RemovalTask* (O_{R2}) that specifies how the finished *Assembly* will be moved out of the workstation. No *HandlingOperations* can yet be defined since the allocation of the *Operations* to *Equipment* has not been made yet. The temporal order between the operations has been defined in accordance with the task decomposition pattern. The same is true for the definition of the initial requirements of each operation.



Operations:

T_{A2}: *AssemblyTask* (<requirements> ($L_{18.1}$), <basePart> ($C_{J.1}$), <hasParts> (V_{S4}))
V_{S4}: *ProcessVariant* (<hasParts> (O_{S4}, O_{A2}, O_{A3}, O_{A4}, O_{A5}, O_{R2}, TR_1, TR_2, TR_3, TR_4, TR_5, TR_6, TR_7))
O_{S4}: *SupplyOperation* (<requirements> ($C_{K.1}$))
O_{A2}: *AssemblyOperation* (<requirements> (L_{18-3}))
O_{A3}: *AssemblyOperation* (<requirements> (L_{18-1}))
O_{A4}: *AssemblyOperation* (<requirements> (L_{18-2}))
O_{R2}: *RemoveOperation* (<requirements> (A {$C_{J.1}$, $C_{J.1}$, $C_{K.1}$, E_{P1}, L_{17}, L_{18}, L_{26}, L_{27}}))
TR_1: *TemporalRelation* (O_{S4}.input1, T_{A2}.input1, after)
TR_2: *TemporalRelation* (O_{A2}.input1, T_{A2}.input2, after)
TR_3: *TemporalRelation* (O_{A2}.input2, O_{S4}.output1, after)
TR_4: *TemporalRelation* (O_{A3}.input1, O_{A2}.output1, after)
TR_5: *TemporalRelation* (O_{A4}.input1, O_{A3}.output1, after)
TR_6: *TemporalRelation* (O_{R2}.input1, O_{A4}.output1, after)
TR_7: *TemporalRelation* (O_{R2}.output1, T_{A2}.output1, before)

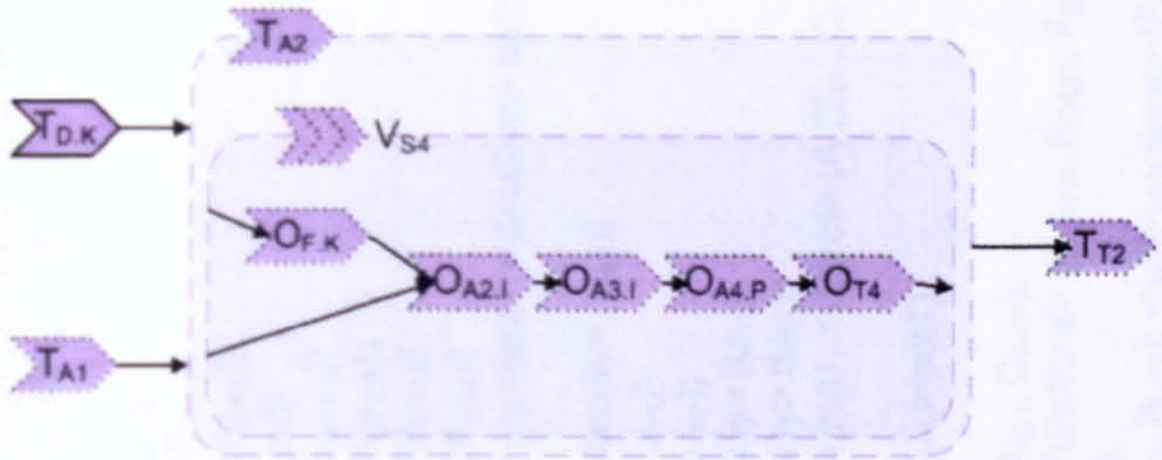
Figure 8.8 Illustration of Riveting Task Decomposition

8.2.1.4 Operation Specialisation

The next step is to choose more specific *Operation* types after an *AssemblyTask* has been decomposed into fundamental types of *Operations*. The specialisation of the *Operations* is taking advantage of the hierarchical classification of the different *Activity* types (see also chapter 7). The *ActivitySpecificationPatterns* are used to maintain the consistence between the chosen *Activity* type and its attribute specification. The <requirements> attribute is of particular importance for the assembly process specification.

The *Operations* of T_{A2} have been specialised as shown in Figure 8.9. The choices have been made based on the requirements defined for the *Operations* during the decomposition in the previous section. The *AssemblyOperations* are specialised based on the *Liaison* type they need to establish. O_{A2} and O_{A3} both establish

LooseFitLiaisons and consequently they should be *InsertionOperations*. O_{A4} establishes a *ContactLiaison* and hence is specialised as *PlaceOperation*.



Specialised Operations:
$O_{F.K}$: *FeedingOperation* (<requirements> ($C_{K.1}$))
$O_{A2.I}$: *InsertionOperation* (<requirements> ($L_{18\text{-}3}$))
$O_{A3.I}$: *InsertionOperation* (<requirements> ($L_{18\text{-}1}$))
$O_{A4.P}$: *PlaceOperation* (<requirements> ($L_{18\text{-}2}$))
O_{T4}: *TransportOperation* (<requirements> (A {$C_{I.1}$, $C_{J.1}$, $C_{K.1}$, E_{P1}, L_{17}, L_{18}, L_{26}, L_{27}}))

Figure 8.9 Illustration of Operation Specialisation in an Assembly Task

The O_{S4} is defining the supply of a *Component* from outside the assembly system boundary and is therefore specified as *FeedingOperation*. The *RemovalOperation* O_{R3} is chosen to be of the type *TransportationOperation* due to fact that the *AssemblyTask* T_{A2} is the last *Task* to be executed by its allocated workstation.

8.2.1.5 Workstation Concept Decomposition (into Units)

A *Workstation* can be broken down into *EquipmentUnit* definitions after the assembly process has been defined up to the operation level. The additional detail generated during the action definition is in the first instance not relevant for the unit level definition of a *Workstation*. The reason is that there is a close relationship between the main *EquipmentUnit* types and the main *Operation* categories. It might become necessary, however, to re-iterate the relationship between *Operations* and *EquipmentUnits* where more than one *Operation* could be allocated to the same *EquipmentUnit*.

The definition of the conceptual equipment specification takes place as a transition from the function, over the behavioural to the structural domain (see chapter 7 for more details). The functional, behavioural, and structural aspects of a higher level conceptual equipment definition can be assigned to lower level ones once they have been defined to a sufficient level of detail. The <hasFunctions>, <hasBehaviour>, and <hasParts> attributes are used to define which aspects belong to a lower level equipment requirements definition.

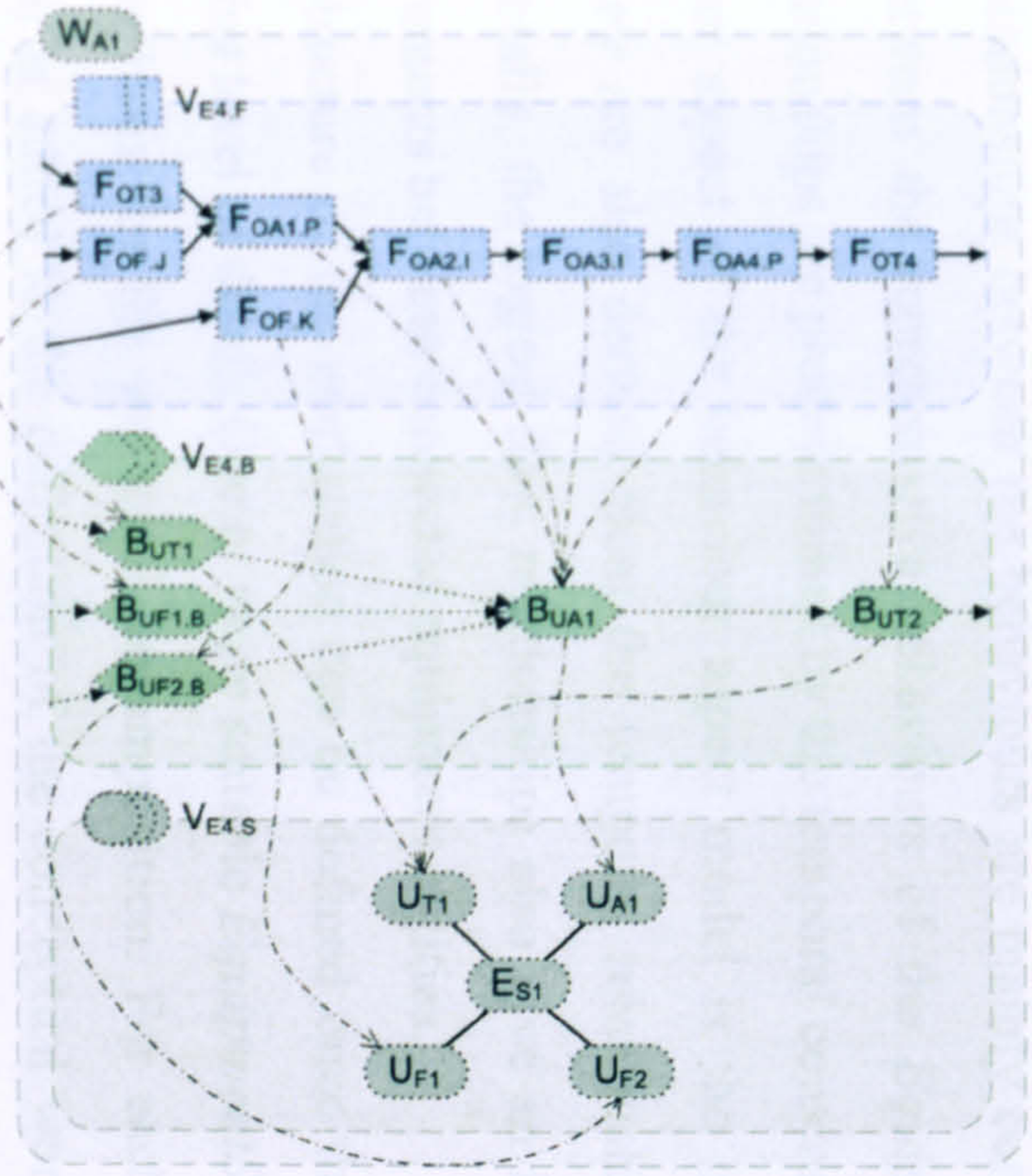


Figure 8.10 Illustration of the conceptual definition of an assembly workstation

Conceptual Equipment Definitions:

W_{A1}: *AssemblyWorkstation*
(<requirements> (T_{A1}, T_{A2}), <hasParts> (V_{E4}, V_{E5}),
<hasBehaviour> (B_{WA1}), <hasFunctions> (F_{TA1}, F_{TA2}))

V_{E4}: *EquipmentVariant*
(<requirements> (V_{S3}, V_{S4}),
<hasFunctions> ($F_{OA1.P}$, $F_{OA2.I}$, $F_{OA3.I}$, $F_{OA4.P}$, $F_{OF.J}$, $F_{OF.K}$, F_{OT3}, F_{OT4}, TR_1, …),
<hasBehaviour> (B_{UA1}, $B_{UF1.B}$, $B_{UF2.B}$, B_{UT1}, B_{UT2}, MF_1, …, IF_1, …),
<hasParts> (U_{A1}, U_{T1}, U_{T2}, U_{F1}, U_{F2}, E_{S1}, C_1, …))

$F_{OA1.P}$: *PlaceOperationFunction* (<achievedBy> (B_{UA1}))
$F_{OA2.I}$: *InsertionOperationFunction* (<achievedBy> (B_{UA1}))
$F_{OA3.I}$: *InsertionOperationFunction* (<achievedBy> (B_{UA1}))
$F_{OA4.P}$: *PlaceOperationFunction* (<achievedBy> (B_{UA1}))
$F_{OF.J}$: *FeedingOperationFunction* (<achievedBy> ($B_{UF1.B}$))
$F_{OF.K}$: *FeedingOperationFunction* (<achievedBy> ($B_{UF2.B}$))
F_{OT3}: *TransportOperationFunction* (<achievedBy> (B_{UT1}))
F_{OT4}: *TransportOperationFunction* (<achievedBy> (B_{UT2}))
TR_1: *TemporalRelation* (<from> F_{TA1}.input1, <to> F_{OT3}.input1, <direction> before)
…
B_{UA1}: *ManipulationAssemblyUnitBehaviour* (<exhibitedBy> (U_{A1}))
$B_{UF1.B}$: *BowlFeederUnitBehaviour* (<exhibitedBy> (U_{F1}))
$B_{UF2.B}$: *BowlFeederUnitBehaviour* (<exhibitedBy> (U_{F2}))
B_{UT1}: *FreeFlowTransportUnitBehaviour* (<exhibitedBy> (U_{T1}))
B_{UT2}: *FreeFlowTransportUnitBehaviour* (<exhibitedBy> (U_{T1}))
MF_1: *MaterialFlow* (<from> B_{WA1}.input1, <to> B_{UT1}.input1, <object> (A {E_{P1}, $C_{I.1}$, L_{26}}))
…
IF_1: *InformationFlow* (<from> B_{UT1}.output1, <to> B_{UA1}.input1)
…
U_{A1}: *AssemblyUnit* (<requirements> ($O_{A1.P}$, $O_{A2.I}$, $O_{A3.I}$, $O_{A4.P}$),
<hasBehaviour> (B_{UA1}), <hasFunctions> ($F_{OA1.P}$, $F_{OA2.I}$, $F_{OA3.I}$, $F_{OA4.P}$))
U_{T1}: *FreeFlowTransportUnit* (<requirements> (O_{T3}),
<hasBehaviour> (B_{UT1}, B_{UT2}), <hasFunctions> (F_{OT3}, F_{OT4}))
U_{F1}: *BowlFeederUnit* (<requirements> ($O_{F.J}$),
<hasBehaviour> ($B_{UF1.B}$), <hasFunctions> ($F_{OF.J}$))
U_{F2}: *BowlFeederUnit* (<requirements> ($O_{F.K}$),
<hasBehavour> ($B_{UF2.B}$), <hasFunctions> ($F_{OF.K}$))
E_{S1}: *StructureElement* ()
C_1: *PhysicalConnection* (<connects> (U_{A1}.port1, E_{S1}.port2))
…

In the given validation example the requirements for the *Workstation* need to be broken down into requirements for its containing *EquipmentUnits*. The first step for the decomposition is to translate the given process based requirements into functions. Figure 8.10 shows an illustration of the decomposition of workstation W_{A1}. The functional model is actually the same as the process model since the relationships between activities and their enabling functions are assumed to be one-to-one for this

work (see chapter 6). Consequently the workstation needs to have the functional capability to carry out two feeding operations, two transport operations, and two types of assembly operations (placing and insertion).

This requires the workstation to have two *FeedingUnitBehaviours* ($B_{UF1.B}$ and $B_{UF2.B}$), one or two *TransportUnitBehaviours* (B_{UT1} and B_{UT2}), and at least one *AssemblyUnitBehaviour* (B_{UA1}). Base on the characteristics of the *FeedingOperation* requirements it can be determined that the components in question should best be fed in a bowl feeder type device. Both *FeedingUnitBehaviours* can consequently be specialised to reflect this. The *TransportUnitBehaviours* can be specialised as free flow type conveyors based on the chosen transport strategy at cell level. The *AssemblyUnitBehaviour* can be specialised as manipulation based since the assembly unit is only going to be responsible for place and insertion type operations. The link between the *Functions* and the *Behaviours* is defined through the <achievedBy> attribute of the *Functions*. During the conceptual equipment definition this relationship is interpreted as a requirement.

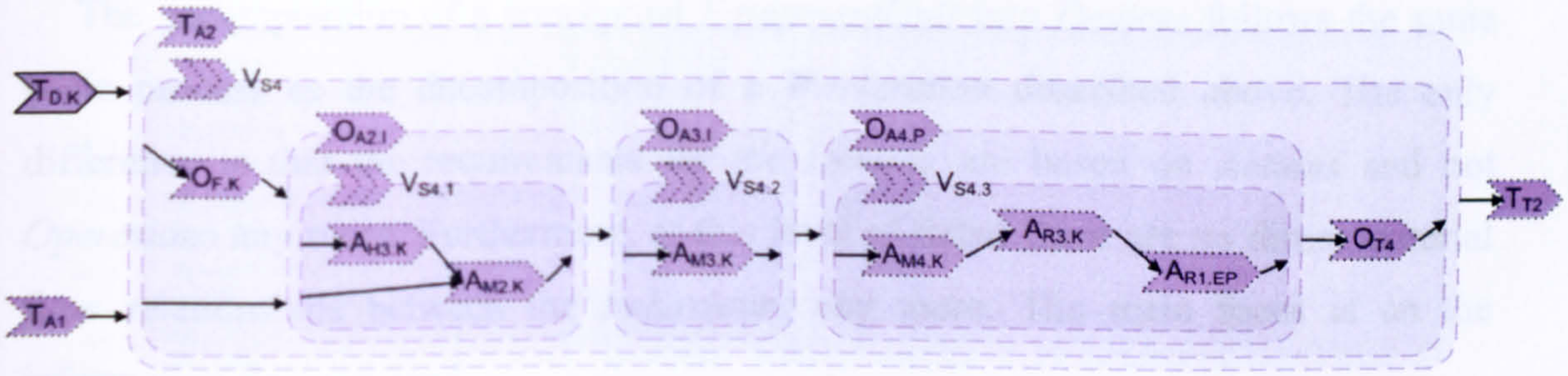
The logical relationships between the behaviours are mainly concerned with the material flow between the representative behaviours of the *EquipmentUnits*. The material flow relationships are predetermined by the temporal constraints between the functions. Another aspect of the behaviour aspect model is the information flow relationships. They are also derived from the temporal relationships between the functions. Incidentally, the logical flow relationships also give an indication of the interfacing requirements between connected equipment entities.

The internal structure of the workstation can be defined once the behaviour has been defined to this level of detail. One or more suitable *EquipmentStructurePatterns* can be chosen as basis for the structural decomposition. For each pattern a new *EquipmentVariant* is added to the definition of the conceptual workstation. In this case a workstation with a central manipulator on a table with an in and out conveyor and the option for a different number of feeders has been chosen. *EquipmentUnits* are created to reflect the required behaviours based on the chosen pattern. The relationship between the *Behaviours* and their enabling conceptual equipment definitions are defined through the <exhibitedBy> attribute of the *Behaviours*. Furthermore, the *Behaviours* as well as the corresponding *Functions* become part of the *EquipmentUnit* definitions. This takes place via the <hasBehaviour> and <hasFunctions> attributes respectively.

The physical connections between the *EquipmentUnits* are defined based on the chosen *EquipmentStructurePattern*. In this case all *EquipmentUnits* are connected to a central table (E_{S1}).

8.2.1.6 Assembly Operation Decomposition and Specialisation

At the action level the direct links to the *Liaison* concept cease to exist. The *Liaison* concept is an abstract representation of the relationships between components and does not have any direct meaning any more at this detailed assembly process specification level. The action level is focused on the actual state transformations that need to take place in the assembly system environment. For example, the movement of objects in space, the gathering of information, the change of process parameters, etc. It is therefore important to keep the relationship to the operation level to be able to understand the actions in the context of assembly. Otherwise it would require sophisticated geometric analysis to recognize that a particular component motion actually establishes a *Liaison*.



Actions:
$O_{A2.I}$: *InsertionOperation* (<requirements> ($L_{18\text{-}3}$), <hasParts> ($V_{S4.1}$))
$V_{S4.1}$: *ProcessVariant* (<hasParts> ($A_{H3.K}$, $A_{M2.K}$, TR_1, ...))
$A_{H3.K}$: *HoldAction* (<requirements> (C_K.P_{K4}))
$A_{M2.K}$: *LinearMoveAction* (<requirements> (C_K))
$O_{A3.I}$: *InsertionOperation* (<requirements> ($L_{18\text{-}1}$), <hasParts> ($V_{S4.2}$))
$V_{S4.2}$: *ProcessVariant* (<hasParts> ($A_{M3.K}$, TR_5, ...))
$A_{M3.K}$: *LinearMoveAction* (<requirements> (C_K))
$O_{A4.P}$: *PlaceOperation* (<requirements> ($L_{18\text{-}2}$), <hasParts> ($V_{S4.3}$))
$V_{S4.3}$: *ProcessVariant* (<hasParts> ($A_{M4.K}$, $A_{R3.K}$, $A_{R1.EP}$, TR_7, ...))
$A_{M4.K}$: *LinearMoveAction* (<requirements> (C_K))
$A_{R3.K}$: *ReleaseAction* (<requirements> (C_K.P_{K4}))
$A_{R1.EP}$: *ReleaseAction* (<requirements> (E_{P1}.P_{EC1}))
TR_1: *TemporalRelation* ($A_{H3.K}$.input1, $O_{A2.I}$.intput1, after)
...

Figure 8.11 Illustrative Action Definition

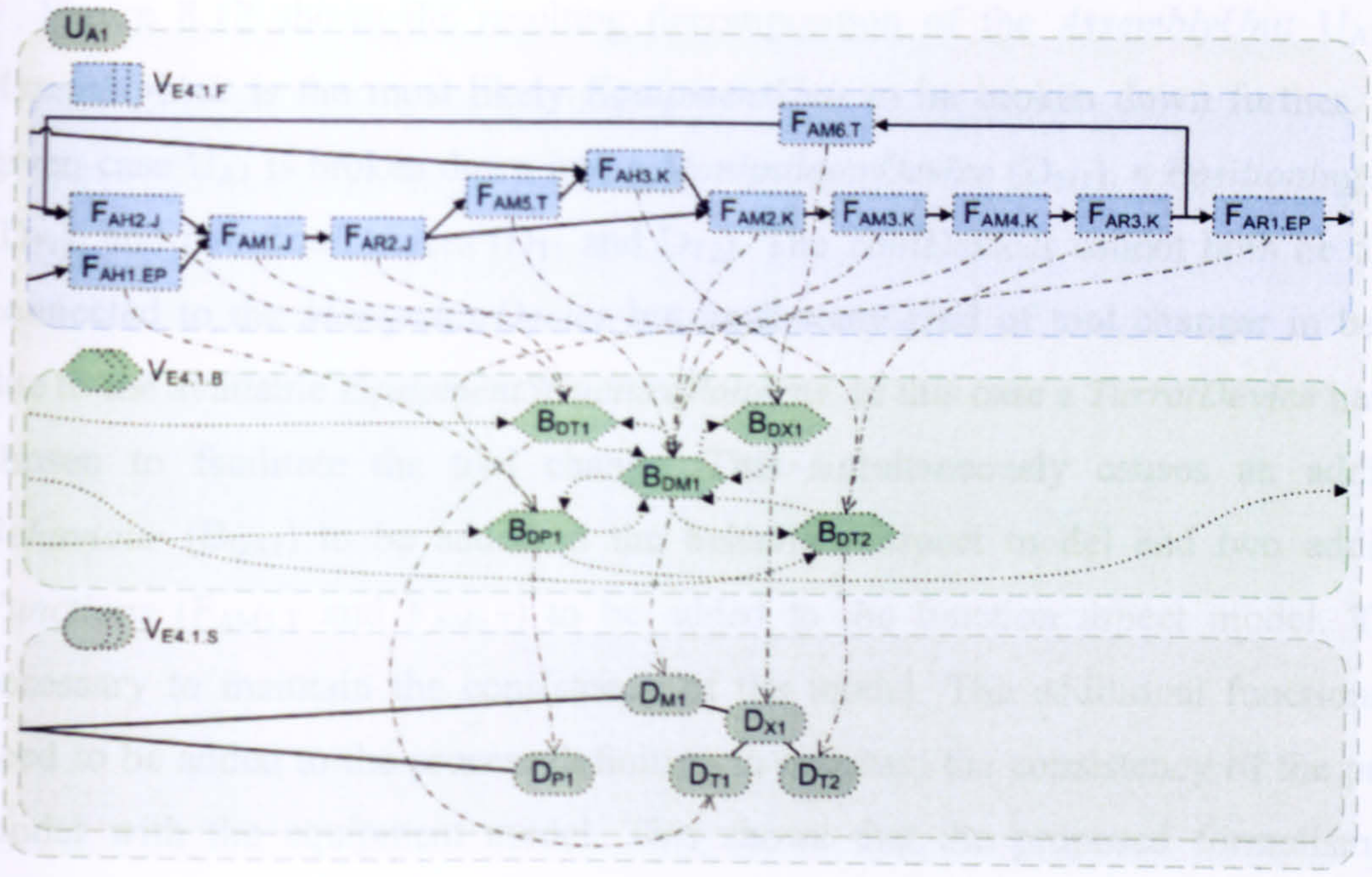
The decomposition of operations into actions and their specialisation uses the same mechanisms as the task decomposition and operation specialisation (see section 8.2.1.3 and 8.2.1.4). Figure 8.11 shows the decomposition and specialisation results of the *AssemblyOperations* in T_{A2}. The three *AssemblyOperations* ($O_{A2.I}$, $O_{A3.I}$, and

$O_{A4.P}$) have a very similar *Action* structure. Basically all three need to contain an enabling *LinearMotionAction*. Additionally they may contain *HoldingActions* and *ReleasingActions* to make the relative motion between the components possible. Both components need to be held before the motion can be carried out. New *HoldingActions*, however, are only required if the components are not held already as the result of preceding assembly processes.

In this specific case the base part is already being held from the previous *AssemblyTask* (T_{A1}) which is carried out in the same workstation. Consequently no additional *HoldingAction* ($A_{H3.K}$) is required as part of any *Operations* in T_{A2}. The *Action* to hold the Rivet (C_{K1}) only needs to be carried out once in the first *AssemblyOperation* ($O_{A2.1}$). Both components need to be released ($A_{R3.K}$ and $A_{R1.EP}$) again at the end of *AssemblyTask* before the resulting assembly can be transported to the next workstation.

8.2.1.7 Unit Concept Decomposition (into Devices)

The decomposition of a conceptual *EquipmentUnit* into *Devices* follows the same basic process as the decomposition of a *Workstation* described above. The only difference is that the requirements for the *Devices* are based on *Actions* and not *Operations* any more. Furthermore, at this level of detail there are no direct material flow relationships between the *Behaviours* any more. The main focus is on the information flows instead.



Conceptual Equipment Definitions:

U_{A1}: *AssemblyUnit*
(<requirements> (O_{A1}, O_{A2}, O_{A3}, O_{A4}),
<hasBehaviour> (B_{UA1}), <hasFunctions> ($F_{OA1.P}$, $F_{OA2.I}$, $F_{OA3.I}$, $F_{OA4.P}$),
<hasParts> ($V_{E4.1}$))

$V_{E4.1}$: *EquipmentVariant*
(<requirements> ($V_{S3.1}$, $V_{S4.1}$, $V_{S4.2}$, $V_{S4.3}$),
<hasFunctions> ($F_{AH1.EP}$, $F_{AR1.EP}$, $F_{AH2.J}$, $F_{AR2.J}$, $F_{AH3.K}$, $F_{AR3.K}$, $F_{AM1.J}$, $F_{AM2.K}$, $F_{AM3.K}$, $F_{AM4.K}$, $F_{AM5.T}$, $F_{AM6.T}$, TR_1, ...)
<hasBehaviour> (B_{DM1}, B_{DT1}, B_{DT2}, B_{DP1}, B_{DX1}, IF_1, ...)
<hasParts> (D_{M1}, D_{T1}, D_{T2}, D_{P1}, D_{X1}, C_1, ...))

$F_{AH1.EP}$: *HoldActionFunction* (<achievedBy> (B_{DP1}), <object> ($E_{P1}.P_{EC1}$))
$F_{AR1.EP}$: *ReleaseActionFunction* (<achievedBy> (B_{DP1}), <object> ($E_{P1}.P_{EC1}$))
$F_{AH2.J}$: *HoldActionFunction* (<achievedBy> (B_{DT1}), <object> ($C_{J.1}.P_{J6}$))
$F_{AR2.J}$: *ReleaseActionFunction* (<achievedBy> (B_{DT1}), <object> ($C_{J.1}.P_{J6}$))
$F_{AH3.K}$: *HoldActionFunction* (<achievedBy> (B_{DT2}), <object> ($C_{K.1}.P_{K4}$))
$F_{AR3.K}$: *ReleaseActionFunction* (<achievedBy> (B_{DT2}), <object> ($C_{K.1}.P_{K4}$))
$F_{AM1.J}$: *LinearMoveActionFunction* (<achievedBy> (B_{DM1}), <object> (D_{T1}))
$F_{AM2.K}$: *LinearMoveActionFunction* (<achievedBy> (B_{DM1}), <object> (D_{T2}))
$F_{AM3.K}$: *LinearMoveActionFunction* (<achievedBy> (B_{DM1}), <object> (D_{T2}))
$F_{AM4.K}$: *LinearMoveActionFunction* (<achievedBy> (B_{DM1}), <object> (D_{T2}))
$F_{AM5.T}$: *RotaryMoveActionFunction* (<achievedBy> (B_{DX1}), <object> (D_{T1}, D_{T2}))
$F_{AM6.T}$: *RotaryMoveActionFunction* (<achievedBy> (B_{DX1}), <object> (D_{T2}, D_{T1}))
TR_1: *TemporalRelation* (<from> F_{OA1}.input1, <to> $F_{AH2.J}$.input1, <direction> before)
...
B_{DM1}: *ManipulatorDeviceBehaviour* (<exhibitedBy> (D_{M1}))
B_{DT1}: *MechanicalGripperDeviceBehaviour* (<exhibitedBy> (D_{T1}))
B_{DT2}: *MechanicalGripperDeviceBehaviour* (<exhibitedBy> (D_{T2}))
D_{DP1}: *LiftPositionDeviceBehaviour* (<exhibitedBy> (D_{P1}))
B_{DX1}: *TurretDeviceBehaviour* (<exhibitedBy> (D_{X1}))
IF_1: *InformationFlow* (<from> B_{UA1}.input1, <to> B_{DT1}.input1)
...
D_{M1}: *ManipulatorDevice* (<requirements> ($A_{M1.J}$, $A_{M2.K}$, $A_{M3.K}$, $A_{M4.K}$),
<hasBehaviour> (B_{DM1}), <hasFunctions> ($F_{AM1.J}$, $F_{AM2.K}$, $F_{AM3.K}$, $F_{AM4.K}$))
D_{T1}: *GrippingDevice* (<requirements> ($A_{H2.J}$, $A_{R2.J}$),
<hasBehaviour> (B_{DT1}), <hasFunctions> ($F_{AH2.J}$, $F_{AR2.J}$))
D_{T2}: *GrippingDevice* (<requirements> ($A_{H3.K}$, $A_{R3.J}$),
<hasBehaviour> (B_{DT2}), <hasFunctions> ($F_{AH3.K}$, $F_{AR3.K}$))
D_{P1}: *PositioningDevice* (<requirements> ($A_{H1.EP}$, $A_{R1.EP}$),
<hasBehaviour> (B_{DP1}), <hasFunctions> ($F_{AH1.EP}$, $F_{AR1.EP}$))
D_{X1}: *TurretDevice* (<requirements> ($A_{M5.T}$, $A_{M6.T}$),
<hasBehaviour> (B_{DX1}), <hasFunctions> ($F_{AM5.T}$, $F_{AM6.T}$))
C_1: *PhysicalConnection* (<connects> (D_{M1}.port2, D_{T1}.port1))
...

Figure 8.12 Illustration of the conceptual definition of an assembly unit

Figure 8.12 shows the resulting decomposition of the *AssemblyUnit* U_{A1}. The *AssemblyUnit* is the most likely *EquipmentUnit* to be broken down further. In the given case U_{A1} is broken down into a *ManipulatorDevice* (D_{M1}), a *PositioningDevice* (D_{P1}), and two *ToolDevices* (D_{T1} and D_{T2}). The *ToolDevices* cannot both be directly connected to the *ManipultorDecice* but need some kind of tool changer in between due to the available *EquipmentStructurePatterns*. In this case a *TurretDevice* has been chosen to facilitate the tool change. This simultaneously causes an additional *Behaviour* (B_{DX1}) to be added to the behaviour aspect model and two additional *Functions* ($F_{AM5.T}$ and $F_{AM6.T}$) to be added to the function aspect model. This is necessary to maintain the consistency of the model. The additional functions also need to be added to the process definition to maintain the consistency of the process model with the equipment model. This shows that the proposed formalisms and mechanisms can be used in both directions to reiterate the definition.

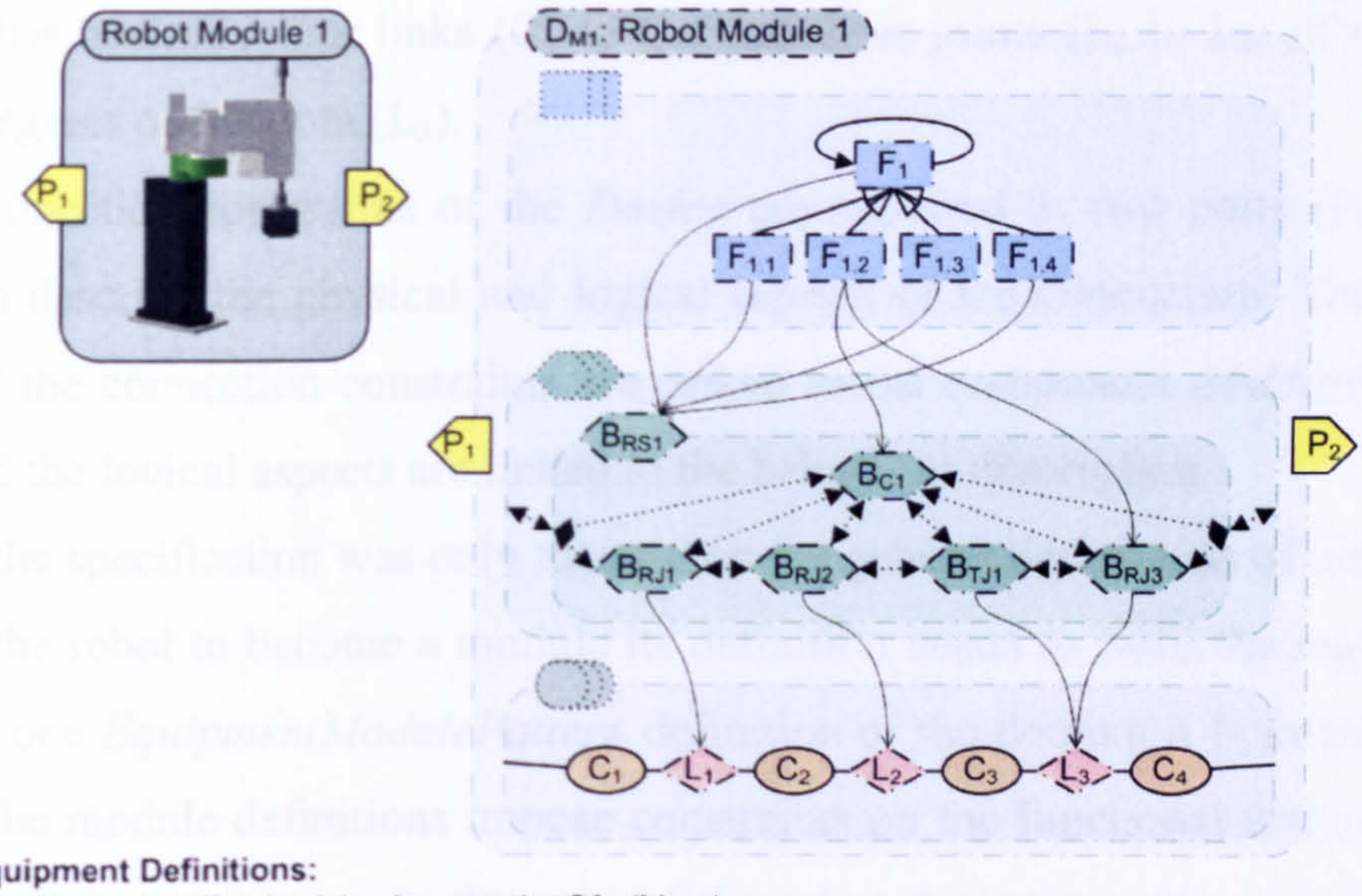
8.2.1.8 Selection of actual equipment modules

Actual equipment modules can be selected as soon as the conceptual definition of a piece of equipment has progressed to a sufficient level. The selection is essentially matching the required equipment characteristics against the characteristics of existing equipment modules (see chapter 7). Consequently the specification of the actual equipment has to address the same aspects as the requirements of the conceptual equipment definition (see chapter 6 for more details).

The definition of the actual available equipment capabilities is naturally more detailed than the definition of the requirements. This is particularly the case for the structure of the equipment but also the behaviour. Figure 8.13 shows the representative specification of a SCARA-type robot (D_{M1}).

The main purpose of the device is to move something which is reflected by its functional capability (F_1). The functional capability of the robot is mapped to the behaviours of the robot that achieve it. Furthermore, the overall move function needs to be defined more specifically to express constraints that become applicable only at the lower level of abstraction. In the case of the robot this is necessary to show that it has only the functional capability to carry out *RotaryMoveActions* ($F_{1.2}$) around its last axis described by B_{RJ3}. The mapping between the functional capabilities of the device and its behaviour needs to include links to two types of behaviours: the technical capability to enable the function and the control capability to carry out the function. In

the case of the main function F_1, it is only linked to the *SCARATypeManipulatorBehaviour* B_{RS1} since it already contains a controller behaviour. Only the *RotaryMoveActionFunction* requires the specification of the controller (B_{C1}) since it is achieved by a lower level technical behaviour (B_{RJ3}).



Equipment Definitions:

D_{M1}: *ManipulatorDevice* (<implementationOf> (M_{DM1}),
<hasFunctions> (F_1, $F_{1.1}$, $F_{1.2}$, $F_{1.3}$, $F_{1.4}$, TR_1),
<hasBehaviour> (B_{RS1}, B_{C1}, IF_1, …, EF_1, …),
<hasStructure> (C_1, C_2, C_3, C_4, L_1, L_2, L_3),
<hasPorts> (P_1, P_2))

F_1: *MoveActionFunction* (<achievedBy> (B_{RS1}))
$F_{1.1}$: *PtPMoveActionFunction* (<specialisationOf> (F_1))
$F_{1.2}$: *RotaryMoveActionFunction* (<specialisationOf> (F_1), <achievedBy> (B_{RJ3}, B_{C1}))
$F_{1.3}$: *LinearMoveActionFunction* (<specialisationOf> (F_1))
$F_{1.4}$: *CircularMoveActionFunction* (<specialisationOf> (F_1))
TR_1: *TemporalConstraint* (<from> F_1.output1, <to> F_1.input1, <direction> before)

B_{RS1}: *SCARATypeManipulatorBehaviour* (<exhibitedBy> (C_1, C_2, C_3, C_4, L_1, L_2, L_3),
<hasParts> (B_{C1}, B_{RJ1}, B_{RJ2}, B_{TJ1}, B_{RJ3}, IF_5, …, EF_5, …))
B_{C1}: *ManipulatorControllerBehaviour* (<exhibitedBy> (D_{M1}), <controlls> (B_{RJ1}, B_{RJ2}, B_{TJ1}, B_{RJ3}))
B_{RJ1}: *RotaryJointBehaviour* (<exhibitedBy> (L_1))
B_{RJ2}: *RotaryJointBehaviour* (<exhibitedBy> (L_2))
B_{TJ1}: *LinearJointBehaviour* (<exhibitedBy> (L_3))
B_{RJ3}: *RotaryJointBehaviour* (<exhibitedBy> (L_3))
IF_1: *InformationFlow* (<from> $D_{M1}.P_1$, <to> B_{RS1}.port1, <object> *Interlog*)
…
EF_1: *EnergyFlow* (<from> $D_{M1}.P_1$, <to> B_{RS1}.port1, <object> Obj_{E230V})
…

C_1: *Component* "Robot Body" (<hasPorts> ($P_{C1\text{-}1}$, $P_{C1\text{-}2}$))
C_2: *Component* "Robot Arm" (<hasPorts> ($P_{C2\text{-}1}$, $P_{C2\text{-}2}$))
C_3: *Component* "Robot Head" (<hasPorts> ($P_{C3\text{-}1}$, $P_{C3\text{-}2}$))
C_4: *Component* "Robot Wrist" (<hasPorts> ($P_{C4\text{-}1}$, $P_{C4\text{-}2}$))
L_1: *JointConnection* (<connects> ($C_1.P_{C1\text{-}2}$, $C_2.P_{C2\text{-}1}$))
L_2: *JointConnection* (<connects> ($C_2.P_{C1\text{-}2}$, $C_3.P_{C2\text{-}1}$))
L_3: *JointConnection* (<connects> ($C_3.P_{C1\text{-}2}$, $C_4.P_{C2\text{-}1}$))

P_1: *EquipmentPort* (<implementationOf> (iP_{M1}), <hasParts> ($P_{C1\text{-}1}$, B_{RS1}.port1))
P_2: *EquipmentPort* (<implementationOf> (iP_{T1}), <hasParts> ($P_{C4\text{-}2}$, B_{RS1}.port2))

Figure 8.13 Illustrative Definition of an SCARA type manipulator device

The behaviour of the robot D_{M1} is defined through a manipulator type composite behaviour which is specialised as SCARA type (B_{RS1}). This behaviour has to contain

three *RotationalJointBehaviours* (B_{RJ1}, B_{RJ2}, and B_{RJ3}), one *TranslationalJointBehaviour* (B_{TJ1}), and a *RobotControllerBehaviour* (B_{C1}). The flows between the behaviours define how they interact to achieve the function.

The structure is also reflecting the basic architecture of a SCARA-type robot. The structure has basically four links (C_1 to C_4) and three joints (L_1 to L_3) of which one has two degrees of freedom (L_3).

The connection constraints of the *Device* are captured in two ports (P_1 and P_2). They both describe the physical and logical aspects of the connection. The physical aspects of the connection constraints are linked to the component description of the device and the logical aspects are linked to the behaviour description.

So far the specification was only focused on the general description of the robot. In order for the robot to become a module its definition needs to fulfil the requirements of at least one *EquipmentModulePattern* definition of the domain it is intended to be used in. The module definitions impose constraints on the functional and connection capability of the equipment. In the case of the robot the most common constraint is placed on its interface ports. The *Device* described above adheres to the specification of a general manipulator type module (M_{DM1}) as depicted in Figure 8.14.

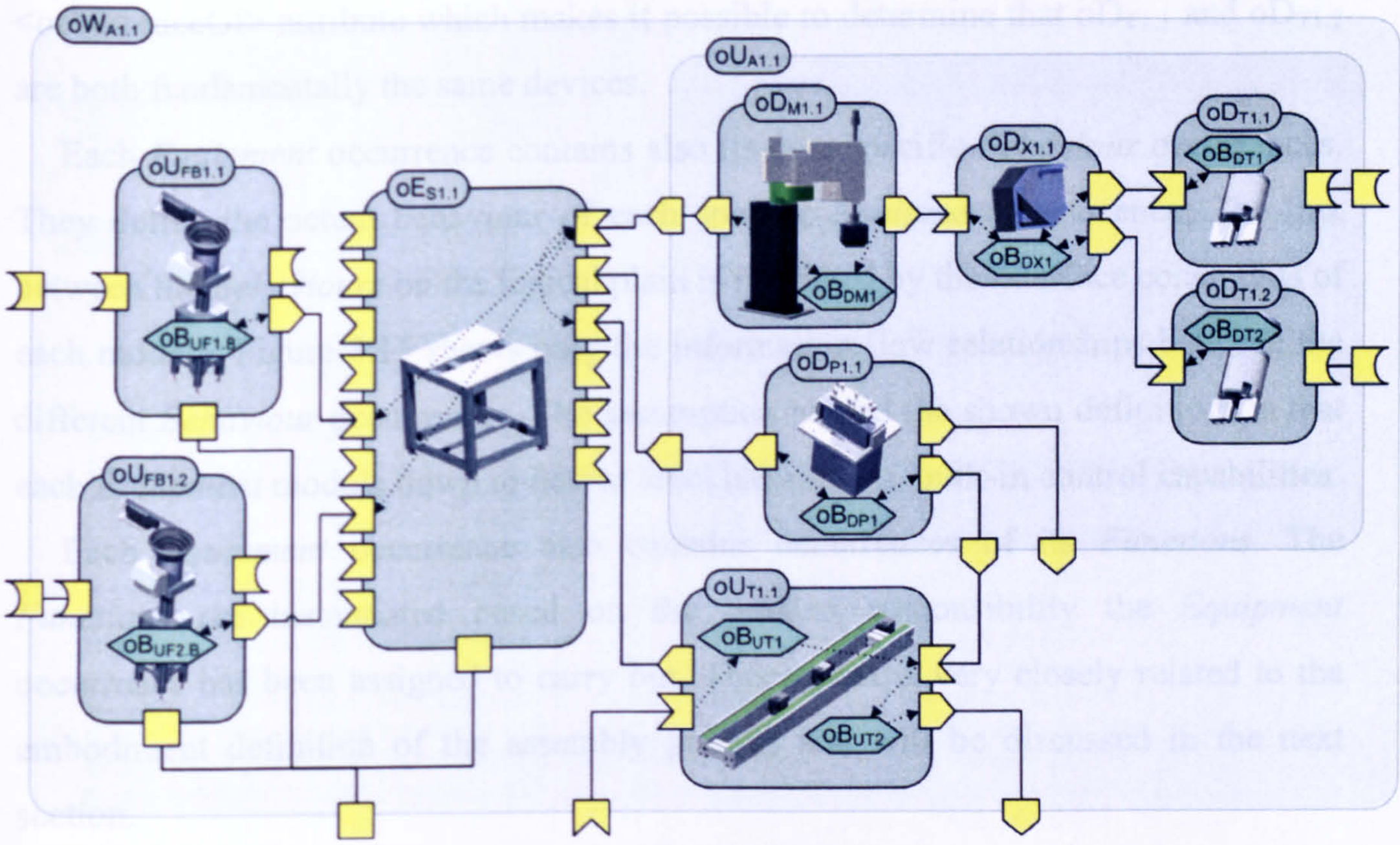
Module Specification:
M_{DM1}: *EquipmentModulePattern*
(<individualType> *ManipulatorDevice*,
<functionConstraints> (F_1, TR_1),
<connectionConstraints> (iP_1, iP_2))
F_1: *FunctionConstraint* (<individualType> *MoveActionFunction*)
TR_1: *TemporalConstraint* (F_1, F_1, before)
iP_1: *InterfacePort* "Table Interface Port"
iP_2: *InterfacePort* "Tool Interface Port"

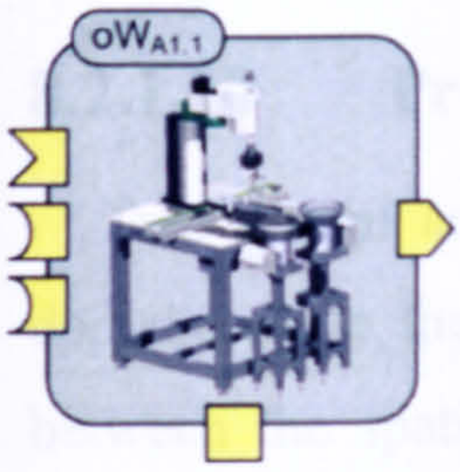
Figure 8.14 Illustrative Module Specification

8.2.1.9 Integration of equipment modules into actual units and workstations

Once at least two *Equipment* modules have been selected, they can be integrated if they actually need to be physically or logically connected. This information is derived from the original conceptual configuration they are part of. Figure 8.15 shows a possible workstation configuration that fulfils the requirements of rW_{A1}. It has the same fundamental structure since no geometric conflicts were detected. $oW_{A1.1}$ is composed of a number of lower level equipment modules. They include: two occurrences of the *BowlFeederUnit* tU_{FB1} ($oU_{FB1.1}$ and $oU_{FB1.2}$), one occurrence of the

FreeFlowConveyorUnit tU_{T1} ($oU_{T1.1}$), one occurrence of the *TableStructureElement* tE_{S1} ($oE_{S1.1}$), and one occurrence of the *AssemblyUnit* configurations tU_{A1} ($oU_{A1.1}$).





Actual Equipment Definitions:

$oW_{A1.1}$: *AssemblyWorkstation* (<occurrenceOf> tW_{A1}, <realisationOf> rW_{A1},
<hasParts> ($oU_{A1.1}$, $oU_{FB1.1}$, $oU_{FB1.2}$, $oU_{T1.1}$, $oE_{S1.1}$, oIC_1, …),
<hasBehaviour> (oB_{WA1}), <hasPorts> ( ))

$oU_{A1.1}$: *AssemblyUnit* (<occurrenceOf> tU_{A1}, <realisationOf> rU_{A1},
<hasParts> ($oD_{M1.1}$, $oD_{P1.1}$, $oD_{X1.1}$, $oD_{T1.1}$, $oD_{T1.2}$, oIC_7, …),
<hasBehaviour> (oB_{UA1}), <hasPorts> ( ))

$oU_{FB1.1}$: *BowlFeederUnit* (<occurrenceOf> tU_{FB1}, <realisationOf> rU_{F1}, <hasParts> (),
<hasBehaviour> ($oB_{UF1.B}$), <hasPorts> ( ))

$oU_{FB1.2}$: *BowlFeederUnit* (<occurrenceOf> tU_{FB1}, <realisationOf> rU_{F2}, <hasParts> (),
<hasBehaviour> ($oB_{UF2.B}$), <hasPorts> ( ))

$oU_{T1.1}$: *FreeFlowConveyorUnit* (<occurrenceOf> tU_{T1}, <realisationOf> rU_{T1}, <hasParts> (),
<hasBehaviour> (oB_{UT1}, oB_{UT2}), <hasPorts> ( ))

$oD_{M1.1}$: *SCARAManipulatorDevice* (<occurrenceOf> tD_{M1}, <realisationOf> rD_{M1}, <hasParts> (),
<hasBehaviour> (oB_{DM1}), <hasPorts> ( ))

$oD_{P1.1}$: *LiftPositionDevice* (<occurrenceOf> tD_{P1}, <realisationOf> rD_{P1}, <hasParts> (),
<hasBehaviour> (oB_{DP1}), <hasPorts> ( ))

$oD_{X1.1}$: *ToolTurretDevice* (<occurrenceOf> tD_{X1}, <realisationOf> rD_{X1}, <hasParts> (),
<hasBehaviour> (oB_{DX1}), <hasPorts> ( ))

$oD_{T1.1}$: *MechanicalGripperDevice* (<occurrenceOf> tD_{T1}, <realisationOf> rD_{T1}, <hasParts> (),
<hasBehaviour> (oB_{DT1}), <hasPorts> ( ))

$oD_{T1.2}$: *MechanicalGripperDevice* (<occurrenceOf> tD_{T1}, <realisationOf> rD_{T2}, <hasParts> (),
<hasBehaviour> (oB_{DT2}), <hasPorts> ( ))

$oE_{S1.1}$:*TableStructureElement* (<occurrenceOf> tE_{S1}, <realisationOf> rE_{S1}, <hasParts> (),
<hasBehaviour> (), <hasPorts> ( ))

oIC_1: *InterfaceConnection* (<connects> (,))

…

Figure 8.15 Actual Workstation Embodiment Specification

The AssemblyUnit aUA1.1 in turn is defined by occurrences of the Devices it is configured from ($oD_{M1.1}$, $oD_{P1.1}$, $oD_{R1.1}$, $oD_{T1.1}$, and $oD_{T1.2}$). The tools are occurrences of the same fundamental tool specification with the only difference that both tool occurrences have been customised for the specific *Component* they are responsible

for. Each actual *Equipment* occurrence is linked to the requirements it fulfils with its <realisationOf> attribute. They are also linked to their generic description via the <occurrenceOf> attribute which makes it possible to determine that $oD_{T1.1}$ and $oD_{T1.2}$ are both fundamentally the same devices.

Each *Equipment* occurrence contains also its own specific *Behaviour* occurrences. They define the actual behaviour of each specific *Equipment* occurrence. The link between the *Behaviours* on the logical plain is restricted by the interface constraints of each module. Figure 8.15 shows only the information flow relationships between the different *Behaviour* occurrences. The assumption behind the shown definitions is that each *Equipment* module down to device level has its own, built-in control capabilities.

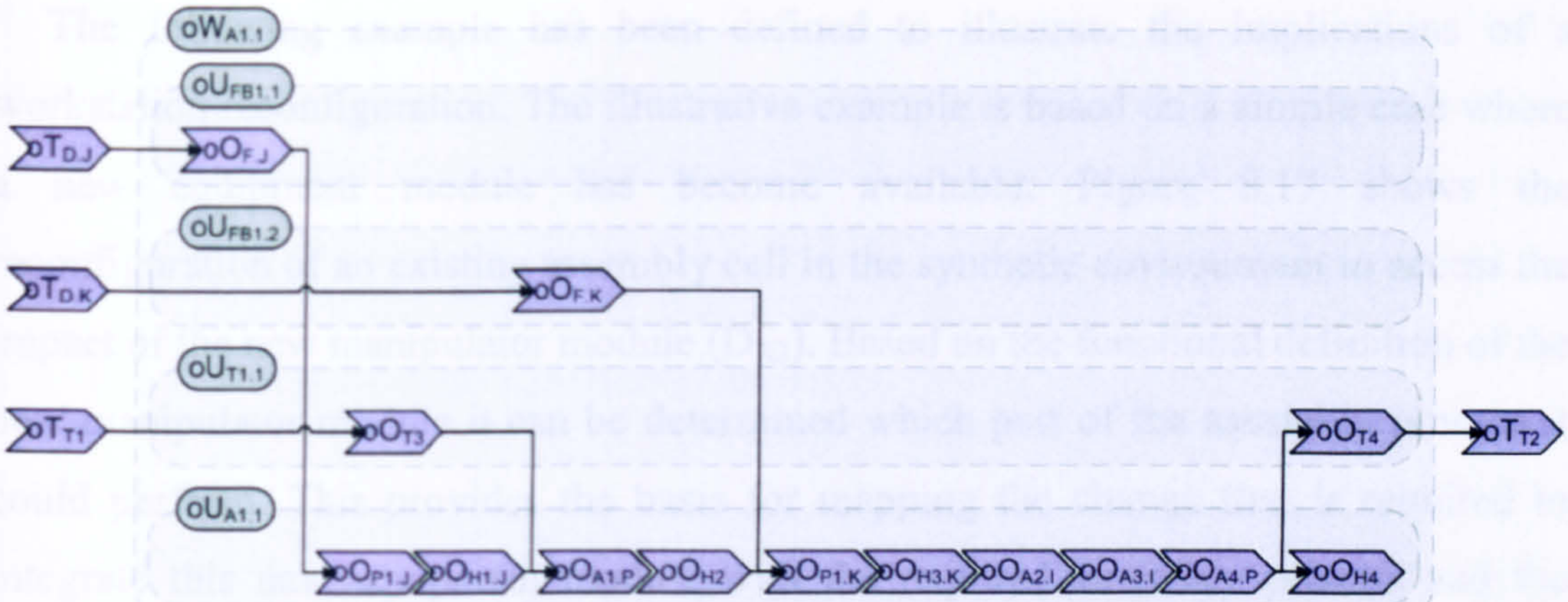
Each *Equipment* occurrence also contains occurrences of its *Functions*. The *Functions* are instantiated based on the process responsibility the *Equipment* occurrence has been assigned to carry out. This aspect is very closely related to the embodiment definition of the assembly process and will be discussed in the next section.

8.2.1.10 Process Model Update

Two design activities need to take place in the assembly process specification domain while the equipment is being configured. One maintains the consistency between the spatial/logical definition of the equipment and the required temporal order of the assembly process specification. The other activity needs to instantiate the actual assembly process embodiment definition based on the functional capabilities of the chosen *Equipment* modules.

In the given example of $oW_{A1.1}$, some additional *HandlingOperations* were required to overcome the spatial difference between the feeding location of the *Components* and their assembly location. Figure 8.16 shows the resulting assembly process embodiment specification for $oW_{A1.1}$ after all constraints have been taken into consideration. Each *Activity* occurrence is linked with the <accomplishedBy> relationship to the *Equipment* occurrences that enable it. They are also linked to the original process requirements that they fulfil. This allows an easy tracking and maintenance of the model.

A similar instantiation process takes place for the embodiment definition of the actual resulting product model. This closes the design loop and the actually achieved product characteristics can be compared back to the originally required ones.



Actual Assembly Process Definition:

$oO_{F.J}$: *FeedingOperation* (<realisationOf> $rO_{F.J}$, <accomplishedBy> ($oU_{FB1.1}$), <responsibleFor> ($oC_{J.1}$))
oO_{T3}: *TransportOperation* (<realisationOf> rO_{T3}, <accomplishedBy> ($oU_{T1.1}$), <responsibleFor> (oA {$oC_{I.1}$, oE_{P1}, oL_{26}}))
$oO_{P1.J}$: *PickUpOperation* (<realisationOf> $rO_{P1.J}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oC_{J.1}$))
$oO_{H1.J}$: *HandlingOperation* (<realisationOf> $rO_{H1.J}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oC_{J.1}$))
$oO_{A1.P}$: *PlaceOperation* (<realisationOf> $rO_{A1.P}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> (oL_{17}, oL_{27}))
oO_{H2}: *HandlingOperation* (<realisationOf> rO_{H2}, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ( ))
$oO_{F.K}$: *FeedingOperation* (<realisationOf> $rO_{F.K}$, <accomplishedBy> ($oU_{FB1.2}$), <responsibleFor> ($oC_{K.1}$))
$oO_{P2.K}$: *PickUpOperation* (<realisationOf> $rO_{P2.K}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oC_{K.1}$))
$oO_{H3.K}$: *HandlingOperation* (<realisationOf> $rO_{H3.K}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oC_{K.1}$))
$oO_{A2.I}$: *InsertionOperation* (<realisationOf> $rO_{A2.I}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oL_{18\text{-}3}$))
$oO_{A3.I}$: *InsertionOperation* (<realisationOf> $rO_{A3.I}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oL_{18\text{-}1}$))
$oO_{A4.P}$: *PlaceOperation* (<realisationOf> $rO_{A4.P}$, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ($oL_{18\text{-}2}$))
oO_{T4}: *TransportOperation* (<realisationOf> rO_{T4}, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> (oA {$oC_{I.1}$, $oC_{J.1}$, $oC_{K.1}$, oE_{P1}, oL_{17}, oL_{18}, oL_{26}, oL_{27}}))
oO_{H4}: *HandlingOperation* (<realisationOf> rO_{H4}, <accomplishedBy> ($oU_{A1.1}$), <responsibleFor> ( ))

Figure 8.16 Assembly Process Embodiment Specification

8.2.2 Reconfiguration of an existing workstation

The proposed approach can also be used to support the adaptation of existing assembly facilities to new and changed requirements. Two potential strategies could be employed to determine the required changes. The first approach would be to design a new assembly system solution from scratch and compare it with the existing system to establish the required changes. This approach, however, could lead to quite excessive and potentially unnecessary changes. The second approach would be to start with the original specification of the existing system and change it until it fits the new requirements. It would be necessary for this approach to understand the original design decisions that led to the specification of the existing system. In this case the actual changes of the original input requirements could be propagated down the decision tree to localise the required changes. Such an approach would allow the minimisation of changes or rather the change effort. The reported framework supports both approaches since it provides a fully constraint model that can be changed and maintained dynamically.

The following example has been defined to illustrate the implications of a workstation reconfiguration. The illustrative example is based on a simple case where a new equipment module has become available. Figure 8.17 shows the reconfiguration of an existing assembly cell in the synthetic environment to access the impact of the new manipulator module (D_{R2}). Based on the functional definition of the new manipulator module it can be determined which part of the assembly process it could perform. This provides the basis for mapping the change that is required to integrate this new equipment module with the required assembly process and the existing system structure.

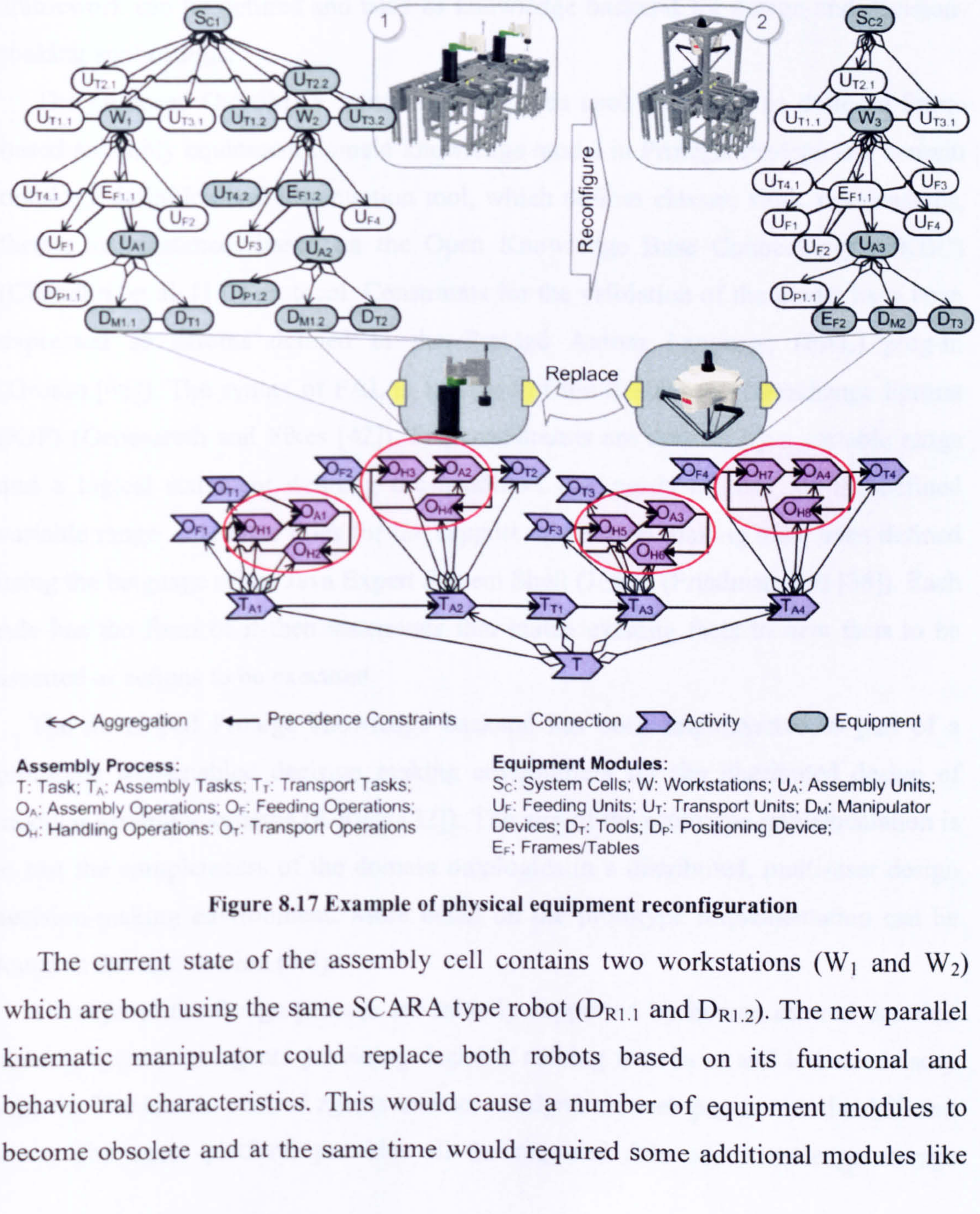


Figure 8.17 Example of physical equipment reconfiguration

The current state of the assembly cell contains two workstations (W_1 and W_2) which are both using the same SCARA type robot ($D_{R1.1}$ and $D_{R1.2}$). The new parallel kinematic manipulator could replace both robots based on its functional and behavioural characteristics. This would cause a number of equipment modules to become obsolete and at the same time would required some additional modules like

an extra frame (E_{F2}) and new tool (D_{T3}). The assembly process definition is simultaneously adapted to the configuration changes. The result is a clear definition of the required changes which is used to assess the trigger conditions for a physical hardware configuration.

8.3 Knowledgebase Development

It is critical that the proposed modular assembly system domain conceptualisation can be translated into practical knowledge based applications. The Protégé framework (Protégé [1]) has been used as part of this work to demonstrate that the ONTOMAS framework can be defined and used as knowledge backend for design and decision-making applications.

The proposed ONTOMAS framework has been used as a basis to define a frame based assembly equipment domain knowledge model in Protégé. Protégé is a domain ontology definition and instantiation tool, which defines classes, slots, relationships, facets, and instances based on the Open Knowledge Base Connectivity (OKBC) (Chaudhri, et al. [17]) protocol. Constraints for the validation of the model have been expressed as axioms defined in the Protégé Axiom Language (PAL) plug-in (Grosso [45]). The syntax of PAL is a variant of the Knowledge Interchange Format (KIF) (Genesereth and Fikes [42]). PAL constraints are defined by a variable range and a logical statement defining the condition that needs to hold for the defined variable range. Inference rules for the support of decision making have been defined using the language of the Java Expert System Shell (JESS) (Friedman-Hill [38]). Each rule has the form of if-then statements that match existing facts to new facts to be asserted or actions to be executed.

The developed Protégé knowledge backend has been implemented as part of a prototype web-enabled decision making environment for the distributed design of modular assembly systems (E-Race [32]). The aim of the prototype implementation is to test the completeness of the domain ontologies in a distributed, multi-user design decision-making environment. More detail on the prototype implementation can be found in the next section (8.4).

All steps of the design process are initially supported by human centred decision making agents, i.e. agents providing decision making interfaces and initial advisory support. The human centred agents interact via dynamic web pages with the different users. The agent platform provides all the required data and knowledge storage

facilities as well as message transport protocols. Figure 8.18 shows the user interface for the embodiment design aspect of an assembly system with the underlying knowledge models defined in Protégé.

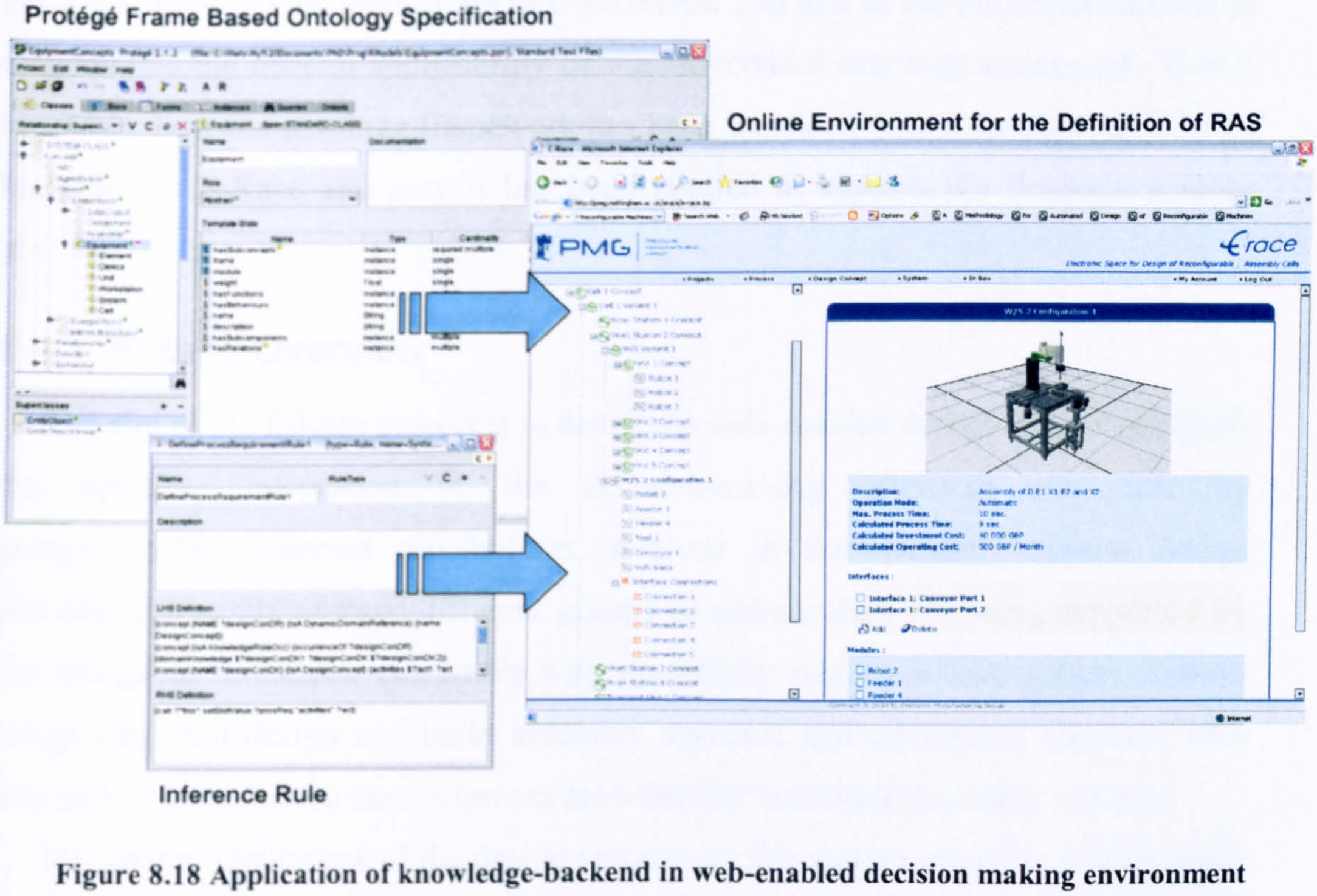


Figure 8.18 Application of knowledge-backend in web-enabled decision making environment

Protégé was also used to test some more intricate axioms and definitions that went beyond the validation of the pure technical applicability of the proposed formalisms towards the creation of a comprehensive domain theory.

8.4 Application in Prototype Environment

The proposed ontology framework ONTOMAS has been used to develop the underlying knowledge backend and decision-making methodology for a web-based assembly system design environment (E-Race [32]). E-Race is a Eureka Factory project funded by the UK department of trade and industry (DTI). The E-Race development is still ongoing and not all aspects of ONTOMAS have been fully integrated yet. However, the applicability of the fundamental ideas and formalisms could be demonstrated.

The developed environment has been rigorously scrutinised by the industrial project partners at several industrial workshops and in one-to-one demonstrations throughout the cause of the project. No major concerns were raised regarding the ontology framework and the underlying methodology reported in this work. On the

contrary, the appropriateness of the framework was generally acknowledged and the potential to move towards a full commercial application has been highlighted.

This section gives an overview of the E-Race web-based design environment with particular focus on the application of ONTOMAS. The aim of the implementation is to demonstrate the general applicability of the ONTOMAS ontology framework. It will be shown how the ontology framework has been translated into a suitable knowledge backend for E-Race and how it has been applied to support the decision making process.

8.4.1 E-Race Overview

The aim of the E-Race project is to develop a web-enabled design space that allows the seamless integration of the decision-making activities conducted by geographically dispersed stakeholders involved in the assembly system design process. The needs of three different groups of stakeholders are being supported by the design environment: customers who potentially require a new system; system integrators who design and build assembly systems; and equipment suppliers who design and manufacture the equipment modules that constitute assembly systems.

The central facilitators of the design process are the system integrators. They have the key task of defining assembly systems that fulfil the requirements of the customer using commercially available equipment modules and solutions from different vendors. This is the most critical task during the assembly system design process dealing with a high level of complexity and requires exhaustive knowledge of the design process as well as the assembly system lifecycle.

8.4.2 General Architecture of the Prototype Environment

The general architecture and application of the E-Race prototype environment was reported in Lohse, et al. [75]. The architecture of the application framework has two aspects: an application server; and a web portal (see Figure 8.19). The application server hosts the decision support environment for the assembly system design process. Each stage of the design process is implemented as independent application modules (agents) that contain their own decision logic, communication facilities and domain knowledge. The application server platform provides a general agent management service, a service directory service, message transport protocols, and communication languages to support the interaction between the decision support agents during the

assembly system design process. The agent platform is defined according to the FIPA abstract architecture for intelligent agents (FIPA SC00001L [35]). The application server allows agents defined in different programming languages (e.g. Java, C++, etc.) to be deployed and participate in the design process. The web portal provides access for the different stakeholders to the relevant decision support agents via a standard web browser.

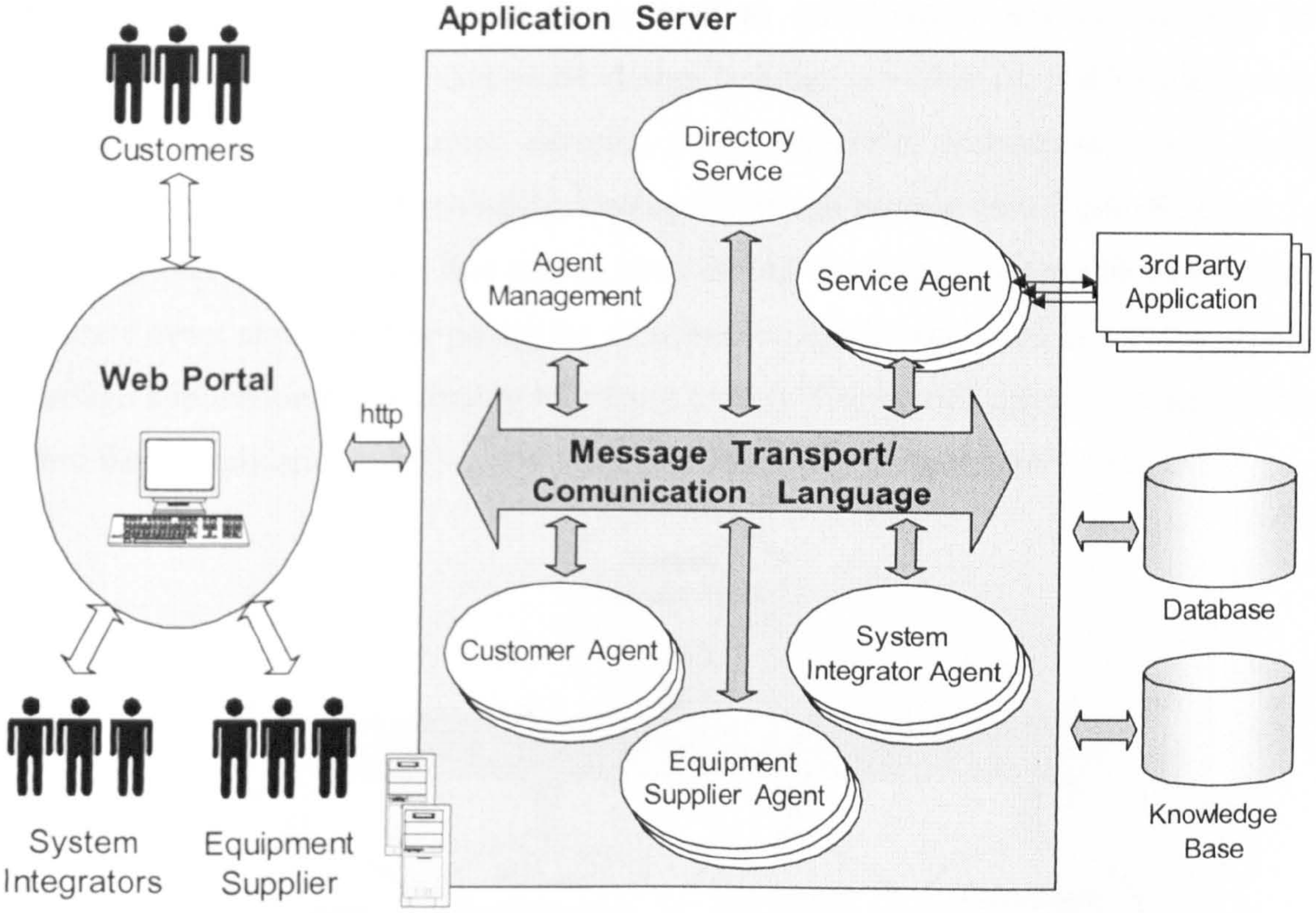


Figure 8.19 Schematic Model of the Assembly System Design Support Framework (Lohse, et al. [75])

The decision support capabilities of the agents can range from simply providing customised user interfaces for decision making to automatic decision making, using various artificial intelligence techniques. Agent can be either directly deployed in agent platform on the main server or on external platforms providing supported message transport protocols and communication languages. The communication between agents running on different server platforms is facilitated via extendable markup language (XML) message exchange.

The agent management system and the service directory service are implemented as autonomous agents. The agent management system agent provides agent lifecycle management and agent white page services listing all deployed agents with their

current lifecycle state. The service directory service agent (yellow page service) lists the services the agents provide and how to utilize them (required languages and ontologies).

The framework deploys a representative agent for each participating stakeholder according to their role (see Figure 8.19). These agents provide basic facilities to manage the design process and the interaction with other stakeholders including sending and receiving of messages as well as initialisation and coordination of relevant design tasks. Each distinctive design task that is within the stakeholder's role is deployed as a separate agent ensuring a high degree of concurrency and a clear separation of the required knowledge during the design process (see Figure 8.20).

Third party applications that can be used during the design process like robot and discrete event simulation or purely for visualisation can be deployed as service agents through application programming interfaces (APIs). The service agents implement the third party applications and register their services on the directory service agent.

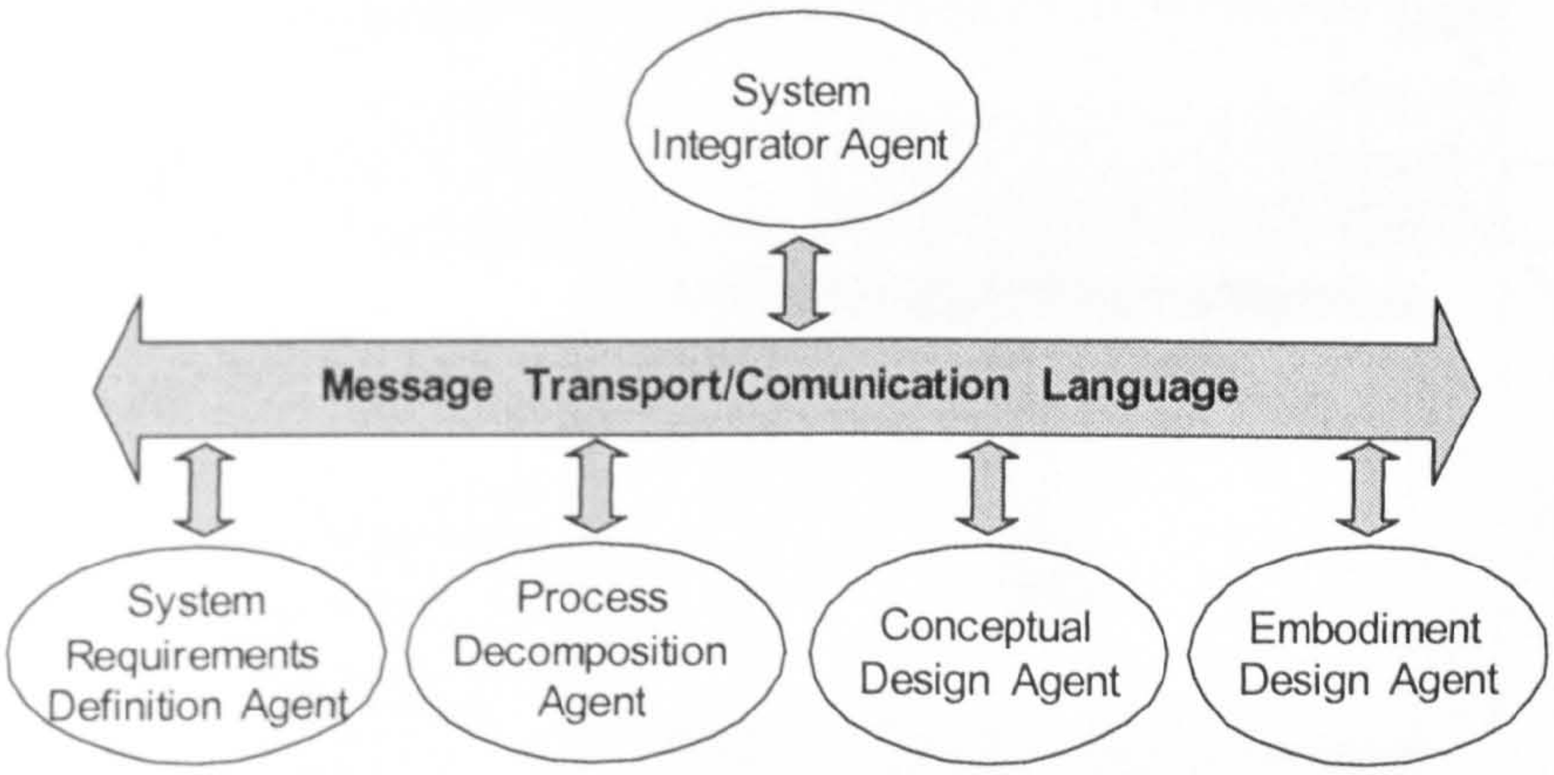


Figure 8.20 Main Design Tasks of a System Integrator (Lohse, et al. [75])

Here, the focus is on the system integrator's role in the design process since they control the design of the assembly system. The decision making activities in the assembly system design process have been grouped into two phases: requirements engineering phase (Hirani [50] and Hirani and Ratchev [51]); and system design phase (Lohse, et al. [70]).

Furthermore, the focus is specifically on the design of assembly workstations using formalised and complete system requirements. However, the decomposition of the process plans, which is traditionally part of requirements engineering, is already constrained by the available equipment solutions. The design of an assembly system is therefore organised as a concurrent process of assembly process decomposition and

system configuration. The configuration is split into two concurrent steps: conceptual and embodiment design. During conceptual design parts of the process plan are grouped into concept entities that define the scope for the equipment selection. In embodiment design equipment modules are selected and configured based on the process scopes of the predefined concepts. During the embodiment design third party programs (service agents) are utilised to ensure the validity of the design and to compare the performance of the resulting system alternatives.

8.4.3 Modular Assembly System Design Process

The fundamental structure of the prototype environment is based on the core formalisms defined by the ONTOMAS framework. The domain experts can define the hierarchical and topological structure of the different domain models and relate them to each other using the cross domain relationships defined by ONTOMAS. The specification of the model is based on the different domain taxonomies.

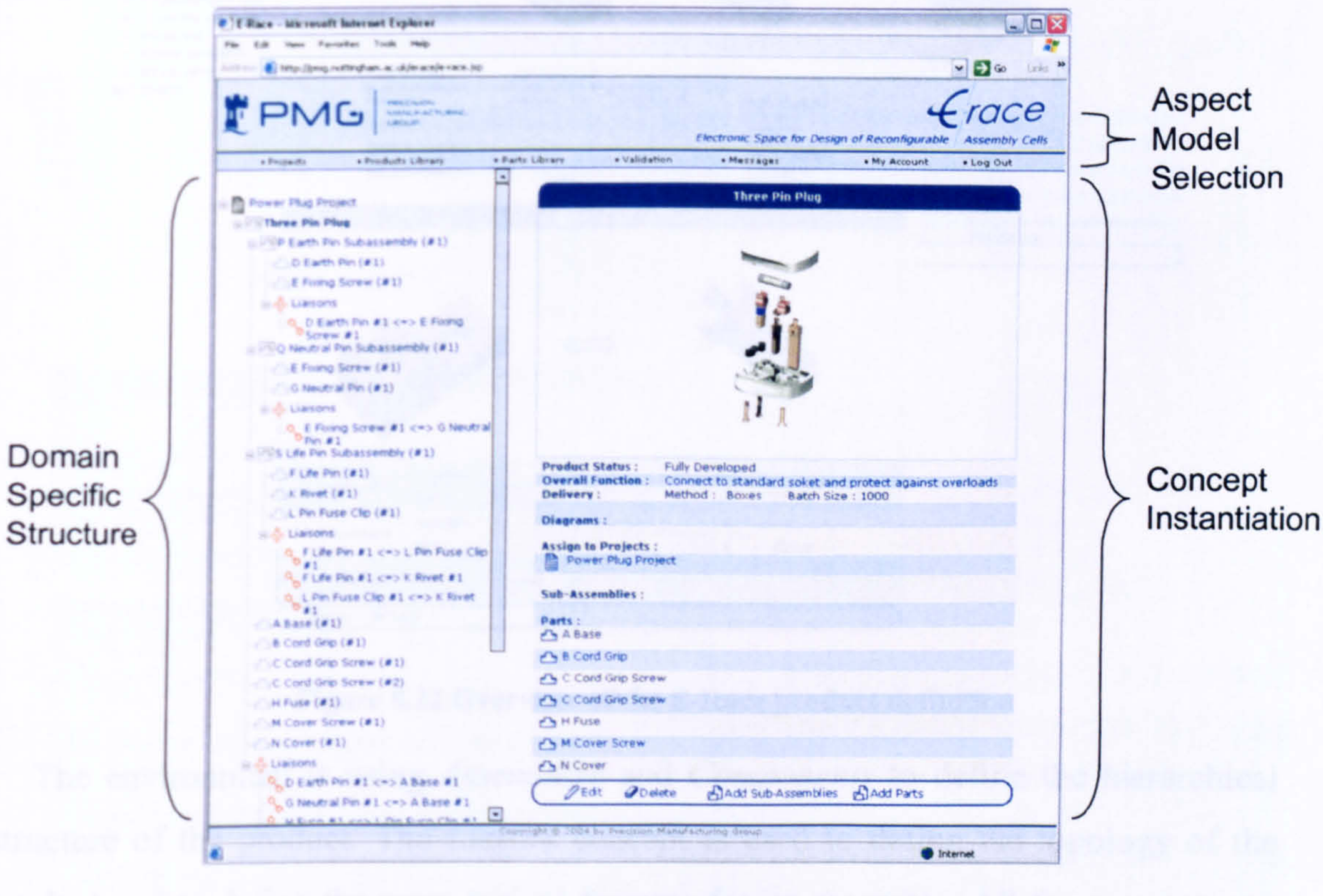


Figure 8.21 Principle layout of the E-Race user interfaces

Figure 8.21 shows an outline of a standard user interface. The domain specific structures are defined on the left side. They reflect the aggregation relationships between the different concept instantiations. At the top is a navigation bar that allows the user to switch between different domain specific views like for example from the product domain view to the assembly process domain view. The characteristics of the

different concepts are defined in the middle. This allows the viewing and specification of the different concept parameters and to integrate it into the topological structure.

In the following sections some exemplary aspects of the prototype design environment will be shown to illustrate how the ONTOMAS formalisms have been applied.

8.4.4 Product Definition

The E-Race prototype environment allows the customer to define their project and product based requirements. The central aspect of the requirements definition is specification of the product. This specification is based on the concepts defined in the product domain ontology of ONTOMAS (see chapter 4).

Figure 8.22 Overview of the E-Race product definition

The environment is using *Assemblies* and *Components* to define the hierarchical structure of the product. The *Liaison* concept is used to define the topology of the product and to define the most critical features for its assembly. All the concepts are linked to the different concept classes defined in the proposed taxonomy structure. Figure 8.22 shows some representative user interfaces from the prototype environment. They illustrate how the different concepts are being utilised.

8.4.5 Assembly Process Definition

The product based requirements are turned into possible assembly processes by the system integrators that have been asked to provide assembly system proposals. The concepts of the assembly process domain ontology of the proposed ONTOMAS framework (see chapter 5) have been applied here to specify the required assembly process. The *Activity* concept is used to define the different required process steps. The <requirements> attribute of the *Activity* concept is implemented to link the process specification to the product based requirements. The *Task*, *Operation*, and *Action* concepts have been applied to hierarchically structure the process specification. Alternative process sequences use the *Variant* aspect of the Activity concept. All *Activity* instances are referenced to the central *Activity* taxonomy. Figure 8.23 shows some representative user interfaces of the assembly process definition. They illustrate how an *AssemblyTask* is being broken down into *AssemblyOperations* and *FeedingOperations*.

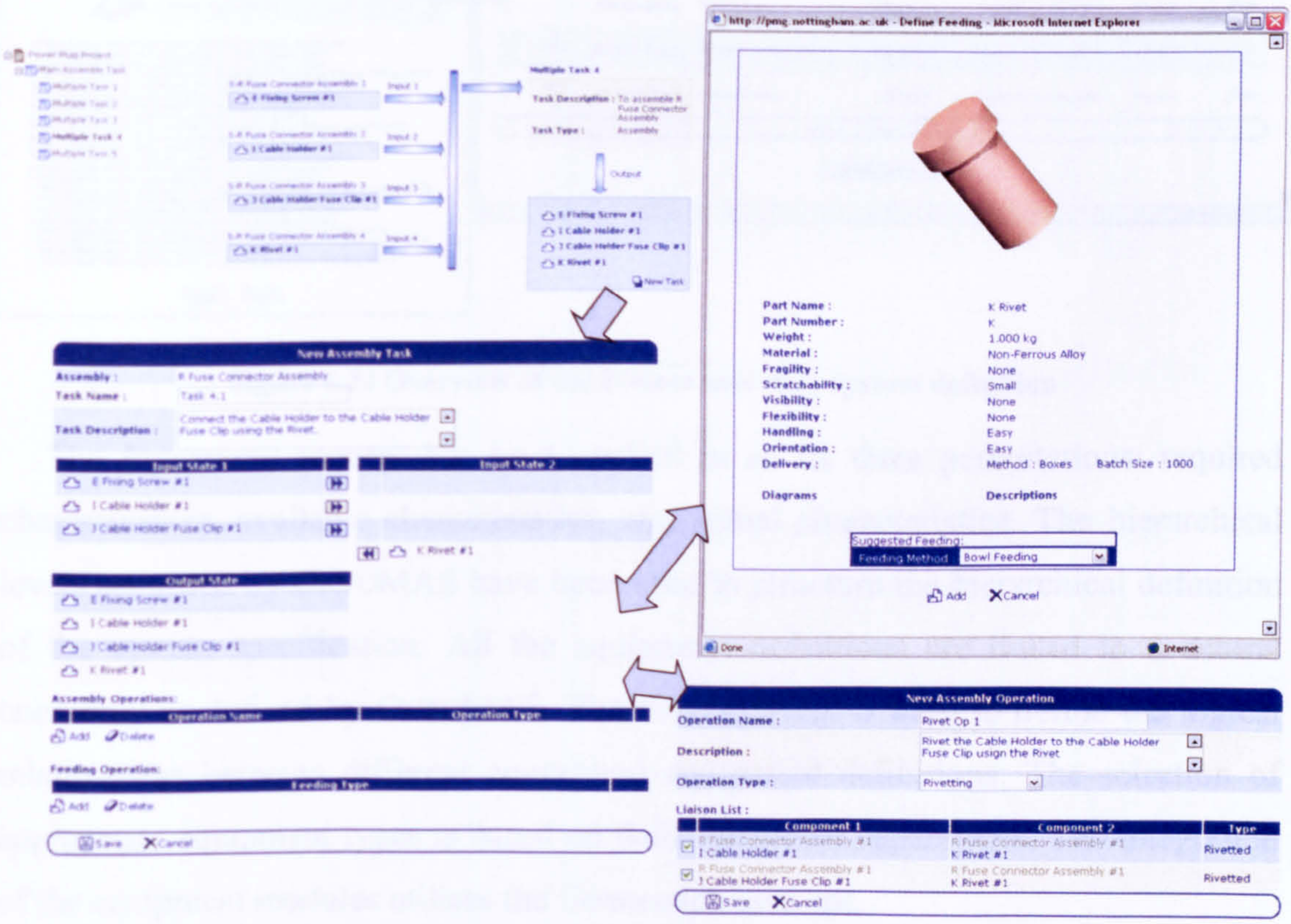


Figure 8.23 Overview of the E-Race assembly process definition

8.4.6 Assembly System Definition

Once the assembly process requirements have been captured, they are turned into equipment requirements by the system integrator. They are then used to find suitable

solutions from the equipment suppliers. The concepts of the ONTOMAS assembly equipment domain ontology defined in chapter 6 have been utilised to specify assembly system solutions.

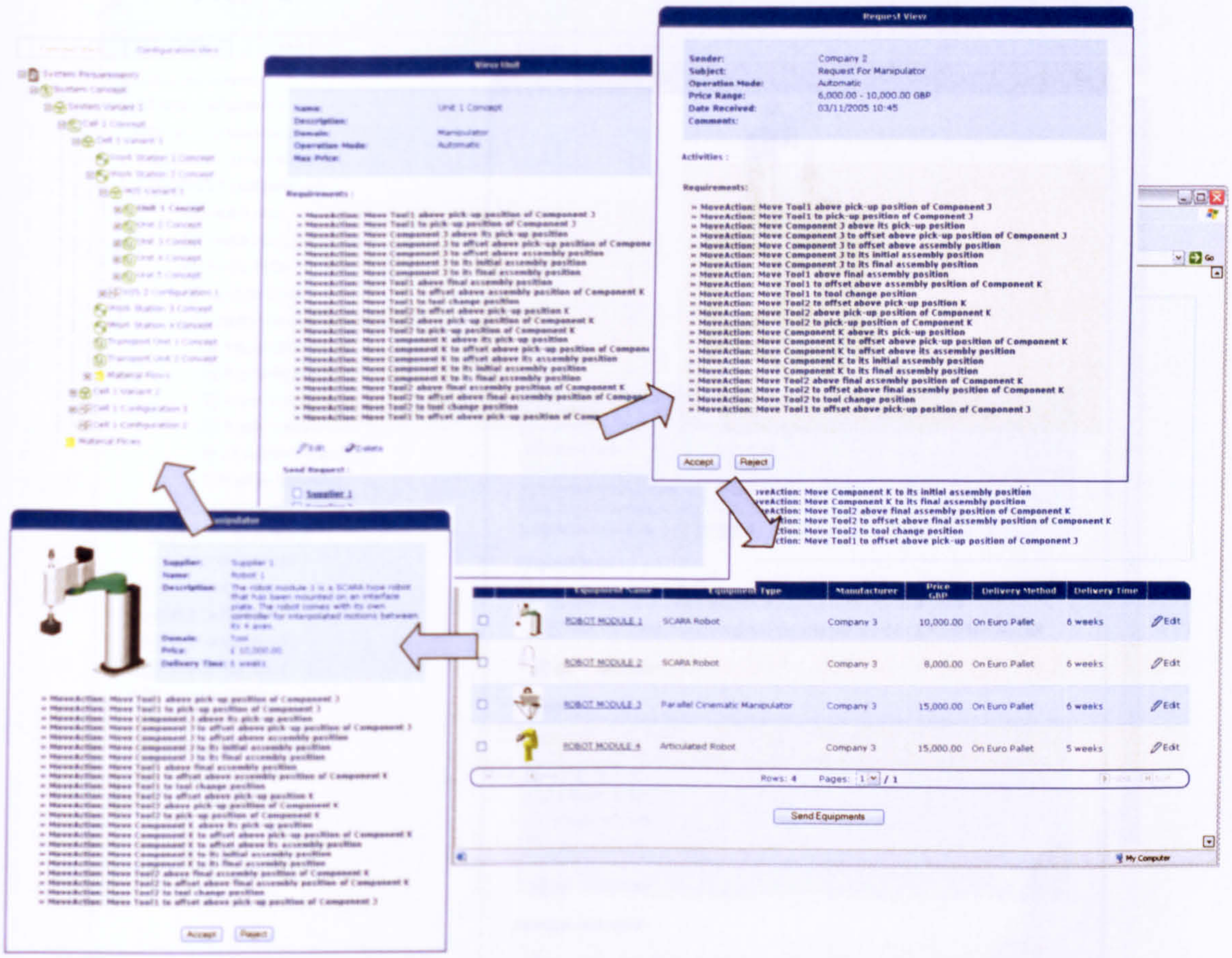


Figure 8.24 Overview of the E-Race assembly system definition

The *Equipment* concept has been applied in all its three permutations: required characteristics, available characteristics, and actual characteristics. The hierarchical levels proposed by ONTOMAS have been used to structure the hierarchical definition of the system specification. All the equipment definitions are linked to a central taxonomy as defined by ONTOMAS. The *Flow* concept is used to define the logical relationships between different conceptual equipment definitions. The selection of appropriate equipment types is based on their functional capabilities. The integration of the equipment modules utilises the *Connection* concept.

Figure 8.24 shows a representative selection of a manipulator device. The selection is starting from the conceptual equipment specification, continues with sending requests to equipment suppliers, waits for the equipment supplier to select a suitable equipment module from his database, and finally integrates the best suitable proposal

with the rest of the equipment. Figure 8.25 shows the result of a workstation configuration.

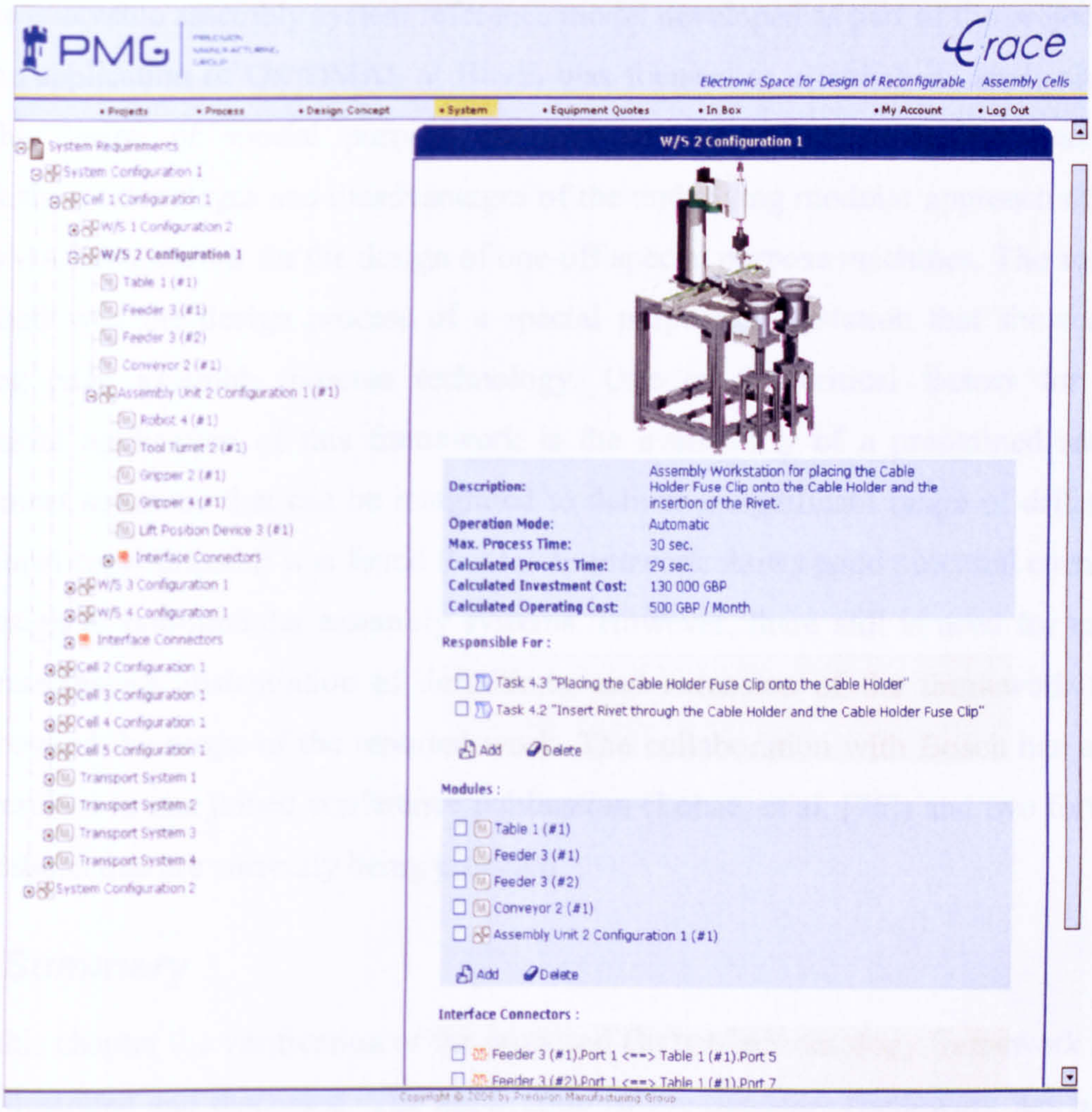


Figure 8.25 Overview of a workstation configuration in E-Race

8.5 Ontology Framework Dissemination and Application

The developed ontology framework has been further scrutinised beyond the use in test cases and a prototype implementation. The proposed ontology is currently being considered as part of the theoretical framework for EUPASS, a major European project (EUPASS [33]). Furthermore the ontology framework has been applied in an industrial project at Bosch Corporate Research in Schwieberdingen to model and analyse a special purpose assembly workstation.

EUPASS is an IP6 integrated framework that addresses the need for Evolvable Ultra-Precision Assembly SystemS. The motivation is to develop assembly system solutions that can be rapidly deployed on demand. A synthetic design environment is considered to be one of the core enabling factors for the success of the project. The

ONTOMAS framework is currently being used to develop the knowledge backend for the EUPASS synthetic environment. Furthermore, it provides one of the key inputs for the evolvable assembly system reference model developed as part of the project.

The application of ONTOMAS at Bosch was focused to establish its applicability for the design of special purpose assembly machines. The emphasis was on establishing advantages and disadvantages of the underlying modular approach of the ONTOMAS framework for the design of one-off special purpose machines. The author has shadowed the design process of a special purpose workstation that showcases leading edge assembly process technology. One of the critical factors for the successful application of this framework is the availability of a predefined set of equipment solutions that can be integrated to deliver a significant range of different functional capabilities. It was found that the framework shows good potential even for the design of non-modular assembly systems. However, there still is need for more elaborate testing, instantiation of definitions, and extension of the framework that goes beyond the scope of the reported work. The collaboration with Bosch has until now resulted in one joined conference publication (Lohse, et al. [76]) and two further publications that are currently being prepared.

8.6 Summary

In this chapter the verification of the proposed ONTOMAS ontology framework has been illustrated and discussed. The application of the ontology framework has been demonstrated with an extensive verification example. All the concepts used in the three different domain models were shown and their application discussed. A case study of a new workstation design and an adaptation of an existing workstation to new equipment modules have been shown.

Furthermore, the implementation and test of the ONTOMAS framework using a widely accepted knowledge backend has been shown. The conceptualisations of the proposed ontology framework were also used as basis for the definition of a prototype assembly system design environment. The implementation in this environment has clearly shown the applicability of the proposed framework.

The key advantages of the proposed ontology framework are that it provides a holistic definition of the assembly system design domain. It defines relationships between different domain aspects, enables computer interpretation through taxonomy formalisms, it addresses the transformation of design information with aspect

definitions of the major concepts, and it provides a clear hierarchy to deal with the inherent complexity of the domain models. Furthermore, the proposed framework provides dynamic formalisms for the utilisation of accumulated design knowledge which can be adapted to changing needs and requirements. Finally, the framework incorporates the principles of modular systems into the domain conceptualisation which allows design decision-making environments to take advantage of the higher level of standardisation. This helps to reduce design time and effort while maintaining a high level of quality.

The results of the verification have so far been very positive. The work has been published in a number of different journals and to several international conferences in the domain (see overview in chapter 3). The proposed ONTOMAS framework was also discussed in detail at a number of industrial workshops and in meetings with industrial partners. Furthermore, the work was applied in an industrial project. No major concerns have been raised by any of the domain experts and only time will tell if the framework will find wider acceptance.

9 Conclusions and Future Work

This thesis set out to contribute to the advancement of knowledge in ways in which ontology frameworks can support the design of modular assembly workstations. The study outlined the state of the art and identified specific requirements for the development of new dynamic design techniques for modular assembly systems. It specifically underlined the limited availability of suitable domain models that could provide a holistic framework at the appropriate level of detail required for integrated design of modular assembly systems. Discussions with domain experts from both industry and academia confirmed the need for a comprehensive modular assembly system domain theory. While the final definition and validation of such a theory is outside the scope of PhD research, this thesis aimed to progress the domain knowledge towards the creation of such a theory by applying new conceptualisation paradigms to the domain of modular assembly systems.

It was hypothesised that domain ontologies based on the following principles would address some of the identified knowledge gaps (see also chapter 3):

- Use of taxonomies to capture the meaning of the different domain concepts
- Definition of fixed hierarchical specification structures to address the inherent complexity of the design problem
- Capturing of reocurring design patterns in predefined concepts
- Application of the function-behaviour-structure paradigm to enable effective equipment selection
- Use of predefined concepts to capture the specific constraints of modular systems

An ontology framework for the design of modular assembly systems (ONTOMAS) has been formulated based on these hypotheses. The developed ONTOMAS framework was applied to several use-cases to verify its applicability. Furthermore, a new design decision making approach was proposed which utilises the ONTOMAS framework for the integrated design and adaptation of modular assembly systems.

9.1 Key Knowledge Contributions

A new integrated ontology framework has been developed to support the design of modular assembly systems (ONTOMAS)

The ONTOMAS framework defines a new set of fundamental modelling formalisms required to capture the specific needs of the knowledge transformations that take place during an assembly system design process. The three key phases of the engineering design process have been captured with an aspect model for each of the core design concepts provided in the ONTOMAS framework. This allows a seamless transformation from required characteristics to actually achieve characteristics based on available capabilities. A small number of basic interrelationships have been identified between the core concepts of the product, process, and equipment domain. This enables a more effective utilisation and maintenance of design constraints throughout the design process. The ONTOMAS framework allows a fully constrained specification of all essential elements of a modular assembly system design process.

A new domain model has been formalised that allows knowledge-enhanced specification of assembly processes.

New fundamental patterns for the specification and decomposition of assembly processes have been formulated as part of the domain model. They allow experts to capture their knowledge about recurring design-decisions in a domain specific, intuitive manner. Three key hierarchical levels in the specification structure, task-level, operation-level, and action-level, have been identified to improve the comprehensiveness of inherently very complex assembly processes. The new assembly process domain model has been fully integrated into the overall ONTOMAS framework. This allows the specification of assembly processes to become an integral part of the wider assembly system design process.

A new equipment domain model has been developed applicable for specification and integration of modular assembly systems.

The equipment domain model is maximising the benefit of modular system solutions. It provides clear function-behaviour-structure aspect models for the

specification of equipment modules. They allow the characteristics of equipment modules to be defined at the right level of detail for effective equipment selection, integration, and evaluation. Similar new modelling formalisms as part of the new assembly process domain model have been developed for the specification of recurring design patterns in the assembly equipment domain. Modular assembly system specific patterns have been identified additionally to the specification and decomposition patterns also identified to assist the specification of processes. A number of key hierarchical levels in the structure of assembly systems have been captured and related to the corresponding levels in the process model. The new assembly equipment domain model is fully integrated in ONTOMAS and allows the specification, selection, and integration of equipment to become an integral part of the overall design process.

Existing product domain models have been extended to capture the specific needs of the design process of modular assembly systems.

New formalisms have been developed to capture the assembly system design relevant characteristics of the product definition. They extend the existing comprehensive knowledge in the area of product modelling by making it more directly accessible. The product domain model has been extended to integrate it into the ONTOMAS framework. This allows the product-driven process requirements specification to become an integrated part of the overall design process.

An integrated design methodology for concurrent assembly process and equipment specification has been developed.

The key decision-making activities during the design and reconfiguration of modular assembly systems have been identified. A new decision-making framework has been defined to enable the iterative, hierarchical design of modular assembly systems across the product, process, and equipment domain. Individual decision-making methods have been specifically designed to maximise the advantage of the ONTOMAS framework while giving domain experts the freedom to utilise their own design strategies.

9.2 Areas of Application

The results of the study are expected to be applicable to a wide range of applications in the modular assembly system design and adaptation domain. The proposed models and methodology will specifically benefit system integrators which have to capture and transform product-based user requirements into suitable assembly system solutions. The key benefit for end users of assembly systems would enable them to access a wider collection of knowledge, and communicate their needs for solutions in a structured way leading to better and more efficient systems solutions. The ONTOMAS framework is expected to be particularly applicable in distributed, multi-vendor environments where there is a specific need for unified product, process and system design specification.

While the primary focus of the research was on modular assembly systems the basic formalisms of the proposed ONTOMAS framework can also be successfully applied to the design of assembly systems with lower levels of modularity where the complex relationships between requirements and equipment solutions need to be efficiently maintained.

The results of the study have already been applied in the European integrated project EUPASS "Evolvable Ultra Precision Assembly Systems" which is an effort of 19 EU industrial and academic partners for development of a new multi-vendor technology platform.

9.3 Future Work

While the reported research addressed a number of issues related to design of modular reconfigurable assembly systems, it also outlined some clear avenues for further research, including:

1. Assembly process taxonomy standardisation needs to be extended over a wide range of different product domains. *For the design approach to be successful it needs to attract a significant number of participants. This however would only be possible if they all speak the same 'language'. The process taxonomy is considered to be the "glue" that connects the product design with the assembly system design. Consequently, this would be the most effective point to start the standardisation of the domain conceptualisation.*

2. A holistic assembly domain theory needs to be developed to further structure and disseminate the knowledge of this domain. *The ONTOMAS framework proposed in this thesis provides the first step towards such a holistic theory. The effort required for the development of a holistic assembly domain theory, however, goes far beyond the scope of an individual PhD research and needs to be the subject of further investigation. A holistic domain theory would allow an easier exchange, consolidation, refinement, and development of existing and new knowledge within the assembly domain.*
3. There is a need for new methods that assist or even automate the derivation of assembly process specification constraints based on the available capabilities of existing equipment module solutions. *In this work it had been assumed that the assembly process specification constraints have been defined. A higher degree of automation would be very beneficial for the derivation of the constraints in an environment where the available equipment solutions are constantly changing. This would enable the constraints to reflect closely the actual possible solutions which could save costly design iterations.*
4. There is a need to apply the proposed assembly process specification to semi-automatically derive control algorithms for modular equipment solutions. *The specification of the assembly process at such high level of detail as defined in this work gives rise to the possibility of using it to derive the required control algorithms for the equipment modules. This would be the case even more so if modular control solutions are used as they would employ inter-logs or message-based interactions to co-ordinate their effort in achieving a common goal. These interactions could be derived from the temporal constraints between the functions for which each equipment module is responsible.*
5. There is a need of further integrating the product design process into a holistic product-process-system design framework. *Currently it is assumed that the product definition is static and is only used to start the requirements specification process. It would be very advantageous to create a link between the product design and its implications on the manufacturing system since most of the cost-driving decisions are already being made during this early stage. This could increase the effectiveness of the proposed framework dramatically.*

9.4 Concluding Remarks

This work started out with the vision to achieve a dynamic integrated design environment for modular assembly systems that would enable the rapid configuration and reconfiguration of assembly solutions on demand. It became obvious that for such a design environment to succeed it would have to be able to integrate a wide range of diverse stakeholders who would be willing to subscribe to common design principles and environments. One of the crucial factors for this to become a reality is the availability of a widely accepted domain conceptualisation that acts as a common language between the different stakeholders. The reported PhD study has introduced an ontology framework for the design and adaptation of modular assembly systems which might provide the future basis for such a domain theory.

While the developed ontology framework is not claiming to be a complete assembly domain ontology, it presents a significant step towards the structuring of this previously very unstructured domain. New domain ontologies for the assembly oriented product, process, and system modelling have been developed as part of this work. They have been integrated into a common ontology framework that addresses the specific needs during the design of modular assembly systems (ONTOMAS). The thesis does not stop at the description of the developed domain ontologies; it also addresses the application of the developed ontology framework in the different design activities during assembly system development.

Despite the significant progress achieved towards the development of an integrated ontology framework for the assembly domain in this work, there still remain major challenges that need to be overcome to create a holistic assembly domain ontology.

References

[1] AAA/Minifactory, (2006), *"An Architecture for Agile Assembly"*, Microdynamic Systems Laboratory, Robotics Institute, Carnegie Mellon University, available at: http://www.msl.ri.cmu.edu/projects/minifactory/

[2] Alsterman, Henric, (2004), *"Strategic Issues for Achieving Sustainable Automatic Assembly - Towards Evolvable Assembly Systems"*, PhD thesis, Royal Institute of Technology, ISSN 1650-1888

[3] Alsterman, Henric, Onori, Mauro, (2001), *"Process-Oriented Assembly System Concept - The Mark IV Approach"*, ISATP 2001, Fukuoka, Japan, May 2001

[4] Amtec, (2006), Amtec Robotics, available at: http://www.powercube.de

[5] Anumba, C. J., Ren, Z., Thorpe, A., Ugwu, O. O., Newnham, L., (2003), *"Negotiation within a multi-agent system for the collaborative design of light industrial buildings"*, Advances in Engineering Software, Elsevier Science Ltd., Vol. 34, pp. 389-401

[6] Barata de Oliveira, José António, (2005), *"Coalition based approach for shop floor agility"*, Edições Orion, ISBN 972-8620-06-3

[7] Bi, Z. M., Zhang, W. J., (2001), *"Modularity Technology in Manufacturing: Taxonomy and Issues"*, The International Journal of Advanced Manufacturing Technology, Springer-Verlag London Ltd., Vol. 18, pp. 381-390

[8] Birmingham, W. P., Brennan, A., Gupta, A. P., Sieworek, D. P., (1988), *"MICON: A single board computer synthesis tool"*, IEEE Circuits and Devices, pp. 37-46

[9] Bley, H., Dietz, S., Roth, N., Zintl, G., (1994), *"Knowledge of Selecting Assembly Cell Components and Its Distribution to CAD and an Expert System for Processing"*, Annals of the CIRP, Vol. 43, No. 1, pp. 5-8

[10] Bock, Conrad, Gruninger, Michael, (2005), *"PSL: A Semantic Domain for Flow Models"*, Software and Systems Modeling, Springer-Verlag GmbH, Vol. 4, No. 2, pp. 209-231

[11] Boër, C. R., Pedrazzoli, P., Sacco, M., Rinaldi, R., De Pascale, G., Avai, A., (2001), *"Integrated Computer Aided Design for Assembly Systems"*, Annals of the CIRP, Vol. 50, No. 1, pp. 17-20

[12] Borst, Pim, Akkermans, Hans, Top, Jan, (1997), *"Engineering Ontologies"*, International Journal of Human-Computer Studies, Vol. 46, pp. 365-406

[13] Bourjault, Alain, (1984), *"Contribution à une approche méthodologique de l'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires"*, Thèse d'Etat, Université de Franche-Comté, Laguage: French

[14] Cencorp, (2006), Cencorp Corporation, available at: http://www.cencorp.com/

[15] Chandrasekaran, B., Josephson, John R., Benjamins, V. Richard, (1999), *"What are ontologies, and why do we need them?"*, IEEE Intelligent Systems, IEEE, pp. 20-26

[16] Chao, Kuo-Ming, Norman, Peter, James, Anne, (2002), *"An agent-based approach to engineering design"*, Computers in Industry, Elsevier Science B. V., Vol. 48, pp. 17-27

[17] Chaudhri, V. K., Farquhar, A., Fikes, Richard E., Karp, P. D., Rice, J. P., (1998), *"Open Knowledge Base Connectivity 2.0.3"*, SRI International, California, USA, available at: www.ai.sri.com/~okbc/

[18] Chen, I-Ming, (2001), *"Rapid response manufacturing through a rapidly reconfigurable robotic workcell"*, Robotics and Computer Integrated Manufacturing, Elsevier Science Ltd., Vol. 17, pp. 199-213

[19] Chen, L., Song, Z., Feng, L., (2003), *"Internet-enabled real-time collaborative assembly modelling via an e-Assembly system: status and promise"*, Computer-Aided Design, Vol. 36, pp. 835-347

[20] Chiang, Chih-Jung, Chirikjian, Gregory S., (2001), *"Modular Robot Motion Planning Using Similarity Metrics"*, Autonomous Robots, Vol. 10, pp. 91-106

[21] Ciocoiu, Mihai, Nau, Dana S., Gruninger, Michael, (2001), *"Ontologies for Integrating Engineering Applications"*, Journal of Computing and Information Science in Engineering, ASME, No. 1, pp. 12-22

[22] Craig, Kevin, De Vito, Mary, Mattice, Mike, La Vigna, Chris, Teolis, Carole, (1999), *"Mechatronic Integration Modeling"*, IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Atlanta, USA, 19-23 September, 1999

[23] Daconta, Michael C., Obrst, Leo J., Smith, Kevin T., (2003), *"The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management"*, Wiley Publishing, Inc., Indiana, Idianapolis, ISBN 0-471-43257-1

[24] DAML, (2006), *"The DARPA Agent Markup Language"*, DARPA, available at: http://www.daml.org

[25] De Lit, Pierre, (2001), *"A comprehensive and integrated approach for the design of a product family and its assembly system"*, PhD Thesis, Université Libre De Bruxelles

[26] Delchambre, Alain, (1996), *"CAD Method for Industrial Assembly: Concurrent Design of Products, Equipment and Control Systems"*, Wiley and Sons Inc., Chichester, West Sussex, UK

[27] Denis, L., Gardan, Y., Perrin, E., (2004), *"A framework for a distributed CAD system"*, Computer-Aided Design, Vol. 36, pp. 761-773

[28] DIN 8593-0, (1985), *"Fertigungsverfahren Fügen - Einordnung, Uterteilung, Begriffe"*, Deutsches Institut für Normung e.V., Beuth Verlag GmbH, Berlin, in German

[29] Dini, G., Bourgeois, F., Chiabra, P., Muth, A., Neri, F., Onori, Mauro, Santochi, M., (2002), *"Assembly Net Taxonomy and Glossary"*, Assembly Net Consortium

[30] Dori, Dov, (2002), *"Object-Process Methodology - A Holistic Systems Paradigm"*, Springer-Verlag, Berlin, Heidelberg, New York, ISBN 3-540-65471-2

[31] eKADS, (2006), *"CommonKADS Knowledge Editor"*, Epistemics, available at: http://www.epistemics.co.uk/ekads

[32] E-Race, (2006), *"E-Space for Collaborative Manufacture of Re-Configurable Assembly Cells"*, EUREKA Factory E!2851, United Kingdom, available at: www.e-race.info

[33] EUPASS, (2006), *"Evolvable Ultra-Precision Assembly SystemS"*, NMP-2-CT-2004-507978, available at: http://www.eupass.org

[34] Farquhar, A., Fikes, Richard E., Rice, J. P., (1997), *"The Ontolingua Server: A tool for collaborative ontology construction"*, International Journal of Human-Computer Studies, Vol. 46, No. 6, pp. 707-727

[35] FIPA SC00001L, (2002), *"FIPA Abstract Architecture Specification "*, Foundation of Intelligent Physical Agents, Geneva, Switzerland

[36] FIPA, *"The Foundation for Intelligent Physical Agents"*, available at: http://www.fipa.org

[37] Frauenfelder, M., (1999), *"Modular robot assembly system for the production of small electrical and medical devices"*, Scandinavian symposium on Robotics, Oule, Finland, 1999

[38] Friedman-Hill, Ernest, (2003), *"Jess in Action - Rule-Based Systems in Java"*, Manning, Greenwich, ISBN 1-930110-89-8

[39] Fujimori, T., (1994), *"Effectiveness of Factory Automation Which Leads to Value Added Manufacturing (Sony's Case Study with Super SMART System)"*, International Symposium on Industrial Robotics, Germany, 1994

[40] Gaugel, Tobias, Bengel, Matthias, Malthan, Dirk, (2004), *"Building a mini-assembly system from a technology construction kit"*, Assembly Automation, Emerald Group Publishing Ltd., Vol. 24, No. 1, pp. 43-48

[41] Gausemeier, Jürgen, Flath, Martin, Möhringer, Stefan, (2001), *"Conceptual Design of Mechatronic Systems Supported by Semi-formal Specification"*, IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Como, Italy, 8-12 July, 2001

[42] Genesereth, Michael R., Fikes, Richard E., (1992), *"Knowledge Interchange Format Version 3.0 Reference Manual"*, Stanford University, Stanford, California, available at: http://www-ksl.stanford.edu/knowledge-sharing/kif/

[43] Giusti, F., Santochi, M., Dini, G., Arioti, A., (1994), *"A Reconfigurable Assembly Cell for Mechanical Products"*, Annals of the CIRP, Vol. 43, No. 1, pp. 1-4

[44] Graves, Stephen C., Lamar, Bruce W., (1983), *"An Integer Programming Procedure for Assembly System Design Problems"*, Operations Research, Operations Research Society of America, Vol. 31, No. 3, pp. 522-545

[45] Grosso, W., (2004), *"PAL Constraint and Queries Tabs"*, available at: protege.standford.edu/plugins/paltabs/PAL_tabs.html

[46] Gruber, Thomas R., (1993), *"A translation approach to portable ontology specifications"*, Knowledge Acquisition, Academic Press Ltd., Vol. 5, pp. 199-220

[47] Grüninger, Michael, (2004), *"Ontology of the Process Specification Language"*, Handbook on Ontologies, edited by: Staab, S., Studer, R., Springer-Verlag, Berlin, Heidelberg, New York, ISBN 3-540-40834-7

[48] Grunninger, M., Lee, J., (2002), *"Ontology: Applications and Design"*, Communications of the ACM, Vol. 45, No. 2, pp. 39-41

[49] Henrioud, Jean-Michel, Bourjault, Alain, (1991), *"LEGA: a computer-aided generator of assembly plans"*, Computer-Aided Mechanical Assembly Planning, edited by: Homem de Mello, L. S., Lee, S., Kulwer Academic Publishers, ISBN 0-7923-9205-1

[50] Hirani, Hitendra, (2004), *"Knowledge Enriched Requirements Specification for Reconfigurable Assembly Systems"*, PhD Thesis, University of Nottingham

[51] Hirani, Hitendra, Ratchev, Svetan, (2002), *"Knowledge Specification For Requirements Engineering of Reconfigurable Assembly Systems"*, 18th International Conference on CAD, CAM, Robotics and Factories of the Future, Porto, Portugal, June 2002

[52] Hirani, Hitendra, Ratchev, Svetan, Lohse, Niels, Valtchanov; George, (2004), *"Web-Based Specification of Reconfigurable Precision Assembly Systems - Industrial Scenarios and Use Cases"*, 2nd International Precision Assembly Seminar, Bad Hofgastein, Austria, February 2004

[53] Hollis, Ralph L., Quaid, Arthur, (1995), *"An Architecture for Agile Assembly"*, Society of Precision Engineering, 10th Annual Meeting, Austin, Texas, USA, October 15-19, 1995

[54] Homem de Mello, Luiz S., Sanderson, Arthur C., (1991), *"A basic algorithm for the generation of mechanical assembly sequences"*, Computer-Aided Mechanical Assembly Planning, edited by: Homem de Mello, L. S., Lee, S., Kulwer Academic Publishers, ISBN 0-7923-9205-1

[55] Homem de Mello, Luiz S., Sanderson, Arthur C., (1991), *"Representation for Assembly Sequences"*, Computer-Aided Mechanical Assembly Planning, edited by: Homem de Mello, L. S., Lee, S., Kulwer Academic Publishers, ISBN 0-7923-9205-1

[56] Husslage, B., van Dam, E., den Hertog, D., Stehouwer, P., Stinstra, E., (2003), *"Collaborative Metamodelling: Coordinating Simulation-based Product Design"*, Concurrent Engineering, Vol. 11, No. 4, pp. 267-278

[57] IEC 61346-1, (1996), *"Industrial systems, installations and equipment and industrial products - Structuring principles and reference designations - Part 1: Basic rules"*, International Electrotechnical Commission, Geneva, Switzerland

[58] IMTI Report, (2000), *"Manufacturing Success in the 21st Century: A Strategic View"*, Integrated Manufacturing Technology Initiative, 16 July 2000, available at: http://www.IMTI21.org

[59] Jacquet, L., Petiot, J. F., Sommer, J. L., Koyama, Y., (1997), *"Analysis of the Setting up of a Concurrent Engineering Scenario"*, Integrated Design and Manufacturing in Mechanical Engineering, edited by: Chedmail, P., Bocquet, J.-C., Dornfeld, D., Kluwer Academic Publishers, ISBN 0-7923-4739-0

[60] Joneja, Ajay, Lee, Neville, (1998), *"A modular, parametric vibratory feeder: a case study for flexible assembly tools for mass customisation"*, IIE Transactions, Vol. 30, pp. 923-931

[61] Kitamura, Yoshinobu, Mizoguchi, Riichiro, (1998), *"Functional Ontology for Functional Understanding"*, Twelfth International Workshop on Qualitative Reasoning, Cape Cod, USA, May 26-29 1998, AAAI Press

[62] Kitamura, Yoshinobu, Mizoguchi, Riichiro, (1999), *"Meta-Functions of Artifacts"*, Thirteenth International Workshop on Qualitative Reasoning, Loch Awe, Scotland, AAAI Press

[63] Kitamura, Yoshinobu, Mizoguchi, Riichiro, (2003), *"Organizing knowledge about functional decomposition"*, International Conference on Engineering Design ICED03, Stockholm, August 19-21 2003

[64] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritchow, G., Van Brussel, H., Ulsoy, A. G., (1999), *"Reconfigurable Manufacturing Systems"*, CIRP Annals, Vol. 48, No. 2

[65] Lastra, José L. M., (2004), *"Reference Mechatronic Architecture for Actor-Based Assembly System"*, thesis, Tampere University of Technology, ISBN 952-15-1210-5

[66] Lenat, D. B., Guha, R. V., (1989), *"Building Large Knowledge-based Systems: representation and inference in the Cyc project"*, Addison-Wesley, Reading, Mass., USA

[67] Levesque, H., Reiter, R., Lesperance, Y., Lin, F., Scherl, R., (1997), *"GOLOG: A logic programming language for dynamic domains"*, Journal of Logic Programming, Vol. 31, pp. 92-128

[68] Lewek, Jörg, (2005), *"Adaptierbares Informationssystem zur Erstellung baukastenbasierter Fertigungseinrichtungen"*, PhD Thesis, Universität Stuttgart, ISBN 3-936947-46-5, Laguage: German

[69] Li, W. D., Ong, S. K., Fuh, J. Y. H., Wong, Y. S., Lu, Y. Q., Nee, A. Y. C., (2004), *"Feature-based design in a distributed and collaborative environment"*, Computer-Aided Design, Vol. 36, pp. 775-792

[70] Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, (2003), *"Task-Based Assembly Planning and Configuration of Assembly Workstations"*, International Symposium on Assembly and Task Planning, Besancon, France, July 2003

[71] Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, (2005), *"Equipment ontology for modular reconfigurable assembly systems"*, CIRP sponsored 3rd International Conference on Reconfigurable Manufacturing, Ann Arbor, MI, USA, 10-12 May, 2005

[72] Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, (2006), *"Equipment ontology for modular reconfigurable assembly systems"*, International Journal of Flexible Manufacturing Systems, Springer, accepted for publication

[73] Lohse, Niels, Hirani, Hitendra, Ratchev, Svetan, Turitto, Michele, (2005), *"An Ontology for the Definition and Validation of Assembly Processes for Evolvable Assembly Systems"*, The 6th IEEE International Symposium on Assembly and Task Planning, Montréal, Canada, July 19-21, 2005

[74] Lohse, Niels, Ratchev, Svetan, Chrisp, Anthony, (2004), *"Function-Behaviour-Structure model for modular assembly equipment"*, International Precision Assembly Seminar IPAS'2004, Bad Hofgastein, Austria, 12-13 February 2004

[75] Lohse, Niels, Ratchev, Svetan, Valtchanov, George, (2004), *"Towards Web-enabled design of modular assembly systems"*, Assembly Automation, Emerald Group Publishing Limited, Vol. 24, No. 3, pp. 270-279

[76] Lohse, Niels, Schäfer, Christian, Ratchev, Svetan, (2006), *"Towards an Integrated Assembly Process Decomposition and Modular Equipment Configuration"*, Precision Assembly Technologies for Mini and Micro Products, edited by: Ratchev, S., Springer, New York, ISBN 0-387-31276-5

[77] Lotter, Bruno, (1990), *"Manufacturing Assembly Handbook"*, Butterworth-Heinemann, ISBN 0408035617

[78] Lu, S. C-Y., Cai, J., Burkett, W., Udwadia, F., (2000), *"A Methodology for Collaborative Design Process and Conflict Analysis"*, Annals of the CIRP, Vol. 49, No. 1, pp. 69-73

[79] McDermott, John, (1982), *"R1: A Rule-Based Configurer of Computer Systems"*, Artificial Intelligence, Vol. 19, pp. 39-88

[80] Meijer, B. R., Tomiyama, T., van der Holst, B. H. A., van der Werff, K., (2003), *"Knowledge Structuring for Function Design"*, Annals of the CIRP, CIRP, Vol. 52, No. 1, pp. 89-92

[81] Merriam-Webster, (2005), *"Merriam-Webster Online Dictionary"*, Merriam-Webster, Inc., available at: http://www.m-w.com

[82] Mervyn, F., Senthil Kumar, A., Bok, S. H., Nee, A.Y.S., (2004), *"Developing distributed applications for integrated product and process design"*, Computer-Aided Design, Vol. 36, pp. 679-689

[83] Mittal, S., Frayman, F., (1989), *"Towards a generic model of configuration tasks"*, Eleventh International Joint Conference on Artificial Intelligence, San Mateo, CA, USA, 1989, Morgan Kaufmann

[84] Mizoguchi, Riichiro, Kitamura, Yoshinobu, (2000), *"Foundation of Knowledge Systematization: Role of Ontological Engineering"*, Industrial Knowledge Management: A Micro-Level Approach, edited by: Roy, R., Springer-Verlag Ltd., London, ISBN 1852333391

[85] Murata, Satoshi, Yoshida, Eiichi, Kurokawa, Haruhisa, Tomita, Kohji, Kokaji, Shigeru, (2001), *"Concept of self-reconfigurable modular robotic system"*, Artificial Intelligence in Engineering, Vol. 15, pp. 383-387

[86] Myon-Woong, Cha, Joo-Heon, Park, Ji-Hyung, Kang, Mujin, (1999), *"Development of an Intelligent Design System for Embodiment Design of Machine Tools"*, Annals of the CIRP, CIRP, Vol. 48, No. 1, pp. 329-332

[87] Nahm, Y., Ishikawa, H., (2004), *"Integrated Product and Process Modeling for Collaborative Design Environment"*, Concurrent Engineering, Vol. 12, No. 1, pp. 5-23

[88] Neville, Dorothy, Joskowicz, Leo, (1992), *"A Representation Language for Conceptual Mechanism Design"*, International Qualitative Reasoning Workshop, June 23, 1992

[89] NGM Project, (1997), *"Next-Generation Manufacturing - A Framework for Action"*, Agility Forum, January 1997, available at: http://www.agilityforum.org

[90] OED, (2003), *"Oxford English Dictionary"*, Oxford University Press, available at: http://www.oed.com

[91] Onori, Mauro, Barata de Oliveira, José António, Lastra, José, Tichem, M., (2002), *"European Precision Assembly Roadmap 2012"*, The Assembly-NET Consortium, November 2002

[92] O'Shea, B., Kaebernick, H., Grewal, S. S., Perlewitz, H., Müller, K., Seliger, G., (1999), *"Method for automatic tool selection for disassembly planning"*, Assembly Automation, Vol. 19, No. 1, pp. 47-54

[93] OWL, (2006), *"OWL Web Ontology Language"*, W3C

[94] Pahl, G., Beitz, W., (1996), *"Engineering Design – A Systematic Approach"*, translated by: Wallace, K., 2nd edition, Springer-Verlag, Berlin Heidelberg New York, ISBN 3-540-19917-9

[95] Paredis, Christiaan J. J., Brown, Benjamin H., Khosla, Pradeep K., (1997), *"A rapidly deployable manipulator system"*, Robotics and Autonomous Systems, Elsevier Science B.V., Vol. 21, pp. 289-304

[96] Pellichero, Fabrice, (2000), *"Computer-aided choice of assembly methods and selection of equipment in production line design"*, PhD Thesis, Université Libre De Bruxelles

[97] Protégé, (2006), *"Protégé, an free, open source ontology editor and knowledge-base framework"*, Stanford University, Stanford University School of Medicine, Stanford Medical Informatics, California, USA, available at: protege.stanford.edu

[98] Qi, Zhiliang, Schäfer, Christian, Klemm, Peter, (2005), *"General Engineering Data Model in Special Purpose Machine Engineering"*, Second International Conference on Informatics in Control, Automation, and Robotics, Barcelona, Spain, September 2005

[99] Rampersad, Hubert K., (1994), *"Integrated and Simultaneous Design for Robotic Assembly"*, John Wiley & Sons Ltd., Chichester, ISBN 0-471-95018-1

[100] Ratchev, Svetan, Hirani, Hitendra, Lohse, Niels, (2004), *"Knowledge Model for the Configuration of Modular Assembly Worksations – Specification and Distributed Conceptual Design"*, 35th International Symposium on Robotics, Paris, France, March 2004

[101] Ratchev, Svetan, Lohse, Niels, (2003), *"Data Modelling for web enabled design of modular precision assembly devices"*, International Precision Assembly Seminar 2003, Bad Hofgastein, Austria, 17-19 March, 2003

[102] Ratchev, Svetan, Lohse, Niels, (2004), *"Data modelling for web enabled design of modular precision assembly devices"*, Assembly Automation, Emerald, Vol. 24, No. 1, pp. 63-70

[103] Ratchev, Svetan, Lohse, Niels, Moualek, Idir, (2002), *"Linguistic based configuration of modular precision assembly devices"*, 33rd International Symposium on Robotics, Stockholm, Sweden, 7-11 October 2002

[104] Ratchev, Svetan, (2000), *"Knowledge-Enriched Requirement Specification for Design of Re-configurable Assembly Cells"*, Delft Workshop on Assembly Automation, Delft, the Netherlands, 11-12 May 2000

[105] Rekiek, Brahim, (2001), *"Assembly Line Design - multiple objective grouping genetic algorithm and the balancing of mixed-model hybrid assembly line"*, PhD Thesis, Université Libre De Bruxelles

[106] RMS Center, (2006), *"NSF Engineering Research Center for Reconfigurable Manufacturing Systems"*, available at: http://erc.engin.umich.edu

[107] Rosenman, Mike A., Gero, John S., (1998), *"Purpose and function in design: from the socio-cultural to the techno-physical"*, Design Studies, Elsevier Science Ltd., Vol. 19, pp. 161-186

[108] Rosenman, Mike A., Gero, John S., (1999), *"Purpose and function in a collaborative CAD environment"*, Reliability Engineering & System Safety, Elsevier Science Ltd., Vol. 64, pp. 167-179

[109] Rosenman, Mike A., Wang, Fujun, (2001), *"A component agent based open CAD system for collaborative design"*, Automation in Construction, Elsevier Science B. V., Vol. 10, pp. 383-397

[110] Roth, Karlheinz, (2000), *"Band 1: Konstruktionslehre"*, 2nd edition, Springer-Verlag, Berlin, language: German, ISBN 3540671420

[111] Sasajima, Munehiko, Kitamura, Yoshinobu, Ikeda, Mitsuru, Mizoguchi, Riichiro, (1995), *"FBRL: A Function and Behavior Representation Language"*, Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), Montreal, Canada, August 21, 1995

[112] Schäfer, Christian, López, Omar, (1999), *"An Object-Oriented Robot Model and its Integration into Flexible Manufacturing Systems"*, Multiple Approaches to Intelligent Systems: 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, edited by: Imam, I. F., Kodratoff, Y., El-Dessouki, A., Ali, M., Springer, ISBN 3540660763

[113] Schlenoff, Craig, Gruninger, Michael, Tissot, Florence, Valois, John, Lubell, Josh, Lee, Jintae, (2000), *"The Process Specification Language (PSL) – Overview and Version 1.0 Specification"*, National Institute of Standards and Technology

[114] Schreiber, Guus, Akkermans, Hans, Anjewierden, Anjo, de Hoog, Robert, Shadbolt, Nigel, Van de Velde, Walter, Wielinga, Bob, (2000), *"Knowledge Engineering and Management - The CommonKADS Methodology"*, The MIT Press, Cambridge, Massachusetts, London, England, ISBN 0-262-19300-0

[115] Seliger, G., Bollmann, O., (1990), *"Knowledge-Based Diagnosis in Flexible Automated Assembly"*, Annals of the CIRP, CIRP, Vol. 39, No. 1, pp. 9-14

[116] Shirinzabeh, Bijan, (1996), *"A CAD-based hierarchical approach to interference detection among fixtgure modules in a reconfigurable fixturing system"*, Robotics & Computer-Integrated Manufacturing, Vol. 12, No. 1, pp. 41-53

[117] Stadzisz, P. C., Henrioud, Jean-Michel, (1998), *"An integrated approach for the design of multi-product assembly systems"*, Computers in Industry, Elsevier Science B. V., Vol. 36, pp. 21-29

[118] Stevens, Richard, Brook, Peter, Jackson, Ken, Arnold, Stuart, (1998), *"System Engineering - Coping with Complexity"*, London, New York, Toronto, Sydney, Tokyo, Singapore, Madrid, Mexico City, Munich, Paris, ISBN 0-13-095085-8

[119] Sugi, M., Maeda, Y., Aiyama, Y., Arai, T., (2001), *"Holonic Robot System: A Flexible Assembly System with High Reconfigurability"*, IEEE International Conference on Robotics and Automation

[120] Suh, Nam P., (1990), "The Principles of Design", Oxford University Press Inc, USA, ISBN 0195043456

[121] Tang, H. P., Wong, T. N., (2005), *"Reactive multi-agent system for assembly cell control"*, Robotics and Computer-Integrated Manufacturing, Vol. 21, pp. 87-89

[122] Tate, Austin, (1998), *"Roots of SPAR - Shared Planning and Activity Representation"*, The Knowledge Engineering Review, edited by: Uschold, M., Tate, A., Cambridge University Press

[123] Tomiyama, Tetsuo, Kiriyama, Takashia, Takeda, Hideaki, Xue, Deye, Yoshikawa, Hiroyuki, (1989), *"Metamodel: A Key to Intelligent CAD Systems"*, Research in Engineering Design, Springer-Verlag Inc., Vol. 1, pp. 19-34

[124] Tomiyama, Tetsuo, Yoshikawa, Hiroyuki, Kiriyama, Takashi, (1993), *"Conceptual Design of Mechanisms: A Qualitative Physics Approach"*, Concurrent Engineering: Automation, Tools, and Techniques, edited by: Kusiak, A., John Wiley & Sons, Inc., ISBN 0-471-55492-8

[125] Travaini, E., Pedrazzoli, P., Rinaldi, R., Boër, C. R., (2002), *"Methodological Approach and Reconfiguration Tool for Assembly Systems"*, Annals of the CIRP, Vol. 51, No. 1, pp. 9-13

[126] Tsukune, H., Tsukamoto, M., Matsushita, T., Tomita, F., Okada, K., Ogasawara, T., Takase, K., Yuba, T., (1993), *"Modular manufacturing"*, Journal of Intelligent Manufacturing, Chapman & Hall, Vol. 4, pp. 163-181

[127] Ulrich, Karl, (1995), *"The role of product architecture in the manufacturing firm"*, Research Policy, Elesevier Science B.V., Vol. 24, pp. 419-440

[128] Ulrich, Karl, Tung, Karen, (1991), *"Fundamentals of Product Modularity"*, Issues in Design Manufacturing/Integration, 1991

[129] Umeda, Yasushi, Ishii, Masaki, Yoshioka, Masaharu, Shimomura, Yoshiki, Tetsuo, Tomiyama, (1996), *"Supporting conceptual design based on the function-behavior-state modeler"*, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Cambridge University Press, Vol. 10, pp. 275-288

[130] Ünsal, Cem, Kiliccöt, Han, Khosla, Pradeep K., (2001), *"A Modular Self-Reconfigurable Bipartite Robotic System: Implementation and Motion Planning"*, Autonomous Robots, Vol. 10, pp. 23-40

[131] VDI 2221, (1993), *"Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte"*, VDI-Verlag, Düsseldorf, in German

[132] VDI 2860, (1990), *"Montage- und Handhabungstechnik - Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole"*, Verein Deutscher Ingenieure, Beuth Verlag GmbH, Berlin, in German

[133] Vos, Jeroen A. W. M., (2001), *"Module and System Design in Flexibly Automated Assembly"*, PhD thesis, Technische Universiteit Delft, ISBN 90-407-2785706

[134] Wang, Ying D., Shen, Weiming, Ghenniwa, Hamada, (2003), *"WebBlow: a Web/agent-based multidisciplinary design optimization environment"*, Computers in Industry, Elsevier B.V., Vol. 52, pp. 17-28

[135] Welch, Richard V., Dixon, John R., (1992), *"Representing function, behavior and structure during conceptual design"*, Design Theory and Methodology, 1992, ASME

[136] Wikipedia, (2006), *"The Free Encyclopedia"*, available at: www.wikipedia.org

[137] WPDL, (2006), The Workflow Management Coalition, available at: http://www.wfmc.org

[138] Xiang, W., Fok, S.C., Thimm, G., (2004), *"Agent-based composable simulation for virtual prototyping of fluid power system"*, Computers in Industry, Vol. 54, pp. 237-251

[139] Yang, Guilin, Chen, I-Ming, (2000), *"Task-based optimization of modular robot configurations: minimized degree-of-freedom approach"*, Mechanism and Machine Theory, Elsevier Science Ltd., Vol. 35, pp. 517-540

[140] Yoshioka, Masaharu, Umeda, Yasushi, Takeda, Hideaki, Shimomura, Yoshiki, Nomaguchi, Yutaka, Tomiyama, Tetsuo, (2004), *"Physical concept ontology for the knowledge intensive engineering framework"*, Advanced Engineering Informatics, Elsevier Ltd., Vol. 18, No. 2, pp. 95-113

[141] Zeigler, Bernard P., Praehofer, Herbert, Kim, Tag Gon, (2000), *"Theory of Modeling and Simulation"*, 2nd edition, Academic Press, London, ISBN 0-12-778455-1

[142] Zha, X. F., Du, H., Lim, Y. E., (2001), *"Knowledge intensive Petri net framework for concurrent intelligent design of automatic assembly systems"*, Robotics and Computer Integrated Manufacturing, Elsevier Science Ltd., Vol. 17, pp. 379-398

[143] Zhang, Mingjun, Fisher, William, Webb, Peter, Tarn, Tzyh-Jong, (2003), *"Functional Model Based Object-Oriented Development Framework for Mechatronic Systems"*, IEEE International Conference on Robotics & Automation, Taipei, Taiwan, 14-19 September, 2003

[144] Zhang, W. J., Liu, S. N., Li, Q., (2000), *"Data/knowledge representation of modular robot and its working environment"*, Robotics and Computer Integrated Manufacturing, Elsevier Science Ltd., Vol. 16, pp. 143-159

[145] Zhang, W., van der Werff, K., (1993), *"A Generic Mechanism Model for Use in a CIM Environment for the Development of Mechanized Production Machines"*, Annals of the CIRP, CIRP, Vol. 42, No. 1, pp. 135-138

[146] Zwegers, Arian, (1998), *"On Systems Architecting – a study in shop floor control to determine architecting concepts and principles"*, PhD thesis, Technische Universiteit Eindhoven, ISBN 90-386-0699-4