

Requirements for a Multidisciplinary Structural-Behavioural Component Model

Dusko Jovanovic

Neopost Technologies

De Tijen 3

9201 BX Drachten, Netherlands

Tel: +31 512 589 300 – Fax: +31 512 589 399 – e-mail: d.jovanovic@neopost.com

Abstract: The reported research responds to a gap in the Product Creation Process, between the subprocesses of Requirements Engineering and System Architecting by means of product modelling. Here an empowerment of the Requirements Engineering (RE) is proposed, by combined structural- and behavioural-wise modelling of multidisciplinary components for embedded systems. The paper motivates and specifies demands for a formalised component model, as modelling notation and workflow, and gives hints for its syntax and semantics; however, besides stating requirements for the semantics, the proposed framework leaves free the choice of a formalism to be applied. It is also explained how this multidisciplinary system-level model interoperates with established monodisciplinary tools and engineering methodologies. An underlying message of this paper is that all relevant structural and behavioural aspects for modelling embedded products at system level can be captured by one generic component model.

Key words: Systems Engineering, multidisciplinary system development, generic behavioural modelling, System Architecting, system decomposition, Requirements Engineering

1. INTRODUCTION

The imperative of sustainability of the western world hi-tech industries in the 21st century is *predictability*, both of a product properties as well as a corresponding project lifecycle. As response to the rapidly developing competitiveness of the eastern world industries (with a strong initial advantage of far less expensive engineering hour), the western industrial practice is increasingly inclining to a full embracement of continuous use of models in the Product Creation Process. With “continuous” is meant here that modelling is not limited to just certain stages of a product development, but that the whole process is covered with uniform interoperable models [10, 2].

A most serious obstacle in attaining continuous modelling are syntactical and semantical disjoints among modelling paradigms in various engineering disciplines involved in creation of a high-tech product. If used at all, discipline-specific models are built in non interoperable semantics, notations and tools.

This, together with the complexity of modern high-tech products, jeopardise sustainability of many advanced industries. As roots of complexity of high-tech products we summarise the following:

1. Features sophistication bordering with intelligent behaviour (*technological*)
2. Multifaceted, highly integrated designs – multidisciplinary (*technological*)
3. Demand for customisation and evolution (*economical*) and answers to it (*technological*)
4. Tight optimisation in balancing among contradicting demands on quality, price and lead time (*economical* - Figure 1).

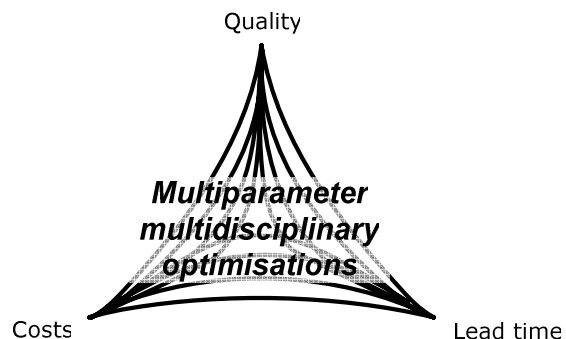


Figure 1 "Devil's triangle" of (embedded) product creation management

This paper is addressing the second - *multidisciplinarity*, and part of the fourth - *quality*, of the quoted sources of complexity. Emphasis is put on links between modelling highly multidisciplinary products and Requirements Engineering, the dominant quality management process.

In this paper a generic, multidisciplinary modelling paradigm is proposed, which integrates structural decomposition, behavioural modelling and quality management. This novel multidisciplinary modelling paradigm is in the sequel referred to as *NeoMultiModel*.

For development of the proposed modelling framework, the industrial domain of “embedded systems” has been aimed (and will be referred in the sequel to as *the domain*). For the sake of clarity, we take “a domain” as a superposition of several engineering disciplines relevant for developing products in a certain industrial sector. Products of the “embedded systems domain” is often referred to as “software-intensive systems” [4]. One characteristic for these systems can be that their interaction is described by finite variables, i.e. localised interfaces.

In the concept of the proposed model, we make no distinction between *discrete* systems – where the interaction can be adequately described by discrete interfaces, and *hybrid* systems – where some interfaces have to include also continuous variables; however, we foresee substantial differences in implementation of the models for discrete and hybrid systems [3], but this issues are left for the time when a concrete syntax and semantics is chosen for implementing the framework, while the modelling paradigm stays invariant.

A substantial empowerment – system’s behaviour predictability – of the Systems Engineering, as one the newest and as such an immature engineering discipline, lies in the heart of this research. The research has been inspired by the success of the component orientation in Software Engineering, which seems to have brought software development to the stage of an “engineering discipline”. The ultimate aim of this research is leveraging the component-wise modelling, as applied in Software Engineering, to multidisciplinary Systems Engineering practice.

2. THE NEED FOR *divide et impera* – SYSTEMS DECOMPOSITION

In dominant product creation models, embedded products have to be developed in projects consisting of many teams (often by subcontracting), which are specialised in particular engineering disciplines, implying concurrent multi-site engineering. Furthermore, in globalised markets with disappearing monopolies, ensuring close realisation of sharp marketing requirements in final product features in a firmly prescribed time span – and all this within tight price limits – is a business survival imperative.

What is described by industry [14] as an urgent need, are highly abstract, yet domain-specific models (i.e. Domain-Specific Languages – DSL’s), based on which it should be possible:

1. to efficiently and precisely *communicate requirements* and *prospective technological solutions* in an early stage of a product conception
2. to efficiently and precisely *define architecture* (i.e. decompose structure and functionality) of the system in terms of well-defined design chunks to be assigned to proper teams or subcontracted to specialised partners
3. to *specify subsystems* with a proper level of detail to let the parts of the systems be concurrently developed within all involved engineering disciplines
4. to *maintain consistency* of the overall architecture upon changes.

3. QUALITY MANAGEMENT = REQUIREMENTS ENGINEERING (RE)

For the meaning of “quality” we take the measure to which extent a product or service comply with expectations [5]. The established engineering discipline that deals with expectations management is widely known as Requirements Engineering (RE).

As a part of the Product Creation Process, RE helps with the aspect of complexity caused by insufficient mutual understanding of stakeholders in product development. RE strives for getting right exact assumptions, requests

and restrictions before a project engages in the engineering phase, and maintain these set of requirements in presence of changes along the project course and in possible project follow-ups.

For this paper goals, the following are taken as the RE objectives:

1. to identify *all relevant properties* of a system to be built
2. to ensure *proper description* of the identified relevant properties
3. to facilitate *prioritising the identified relevant properties* by all the stakeholders by proper understanding of provided descriptions
4. to ensure *completeness and consistency* of these requirements, also when they change during the system lifetime
5. to provide *input for verification and validation* processes.

The deliverable of the RE process are traditionally *requirements specification* (“req-specs”) documents. The RE process is often organised according to the well known V-model, of which one instance is given in Figure 2, that features four req-specs documents. These are:

- StRS – Stakeholder Requirements Specification
- SRS – System Requirements Specification
- SDS – System Design Specification
- CRS – Component Requirements Specification.

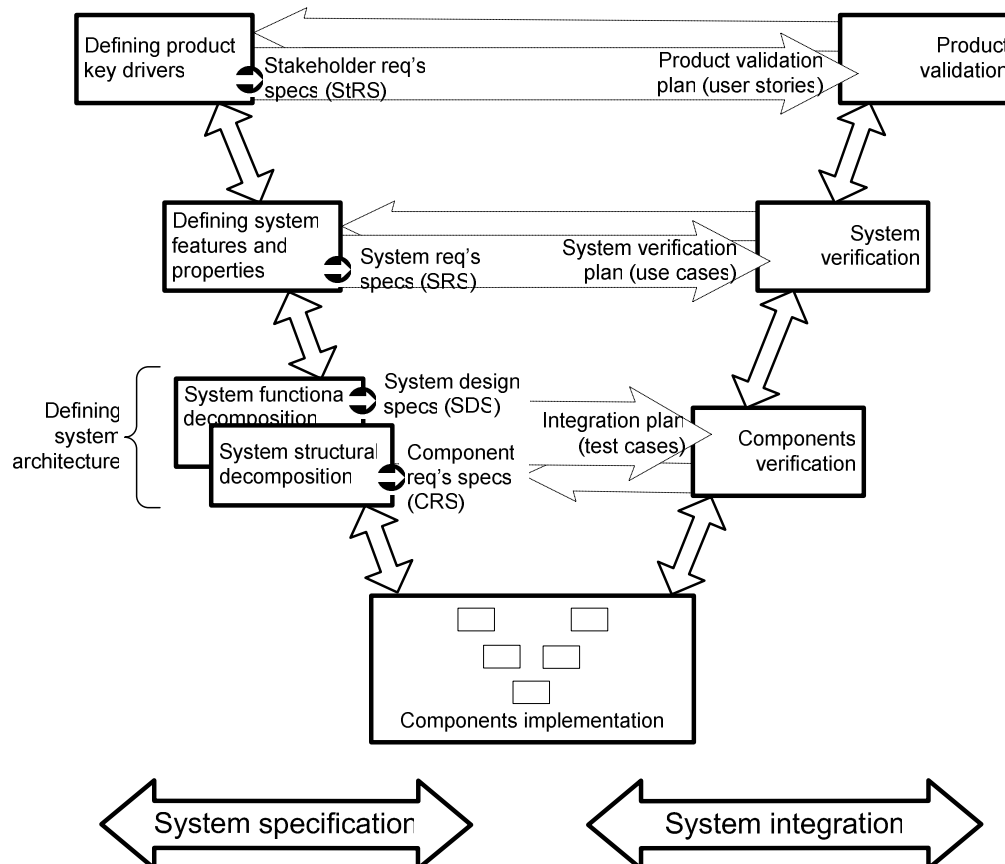


Figure 2 A V-model with emphasized project phases dominated by the RE process

Briefly, system information is distributed over these documents in the following way:

- StRS: description of the features of the system observed as a black box; the language is at a managerial level (user stories); the document answers the question “*what* one wants to achieve with the product”
- SRS: system’s properties still described from the outside, but in quantifiable, technical, domain-specific terms (system use cases); the document answers the question “*what* the system will provide”
- SDS: renders an architecture which decomposes system requirements in external and internal functionalities, with indications of the existing domain knowledge and (reused) components; the document answers the question “*how* the system provides what is required”
- CRS: on basis of the structured decomposition in components indicated in the SDS, specifies contractual interfaces of the system’s components above a certain hierarchical deepness; the CRS documents answer “*what* a defined component will provide”.

Some examples of the tools widely used to equip a RE process are RequisitePro/ClearCase of IBM Rational, DOORS of Telelogic (recently also acquired by IBM), or OSRMT – Open Source Requirements Management Tool – from the open source community.

The state of the art of the RE processes and tools in supporting the *left wing* of the “V” in Figure 2 are means for facilitating systematic authoring the RE documents and ensuring their mutual consistency. The RE documents rely on textual and possibly pictorial specifications of requirements, but in general, for complex products, are often experienced as bureaucratic, and detached from the product *creation* process. Moreover, traditionally without an up-to-date connection with the models of the product, no guarantee can be given that “proper description” of the abovementioned objective 2, or “completeness and consistency” of the abovementioned objective 4, are ensured till the very end of a project.

The *right wing* to the “V” – verification and validation project activities – establishes test specifications based on the RE documents (req-specs) created in the left wing. When the req-specs documents lack formality, their interpretation can be more or less arbitrary. The proposed specifying/modelling framework strives for allowing the highest level of formalisation – mathematical rigorousness – in models supporting specifications to establish a strongest possible link between the specification and verification (integration) processes.

4. DEALING WITH MULTIDISCIPLINARITY

Difficulties in communication among engineers with backgrounds in different disciplines are well known. The modelling approach proposed in this research seeks a language *generic* enough to allow people from disparate disciplines to communicate in a qualitative but also quantitative manner, however *expressive* enough to capture all the relevant aspects of an embedded system for later discipline-specific refinements. This balance between genericness and expressiveness determines also applicability in the domain of the proposed modelling framework.

The choice of notions and notations capable to fulfil the need is scarce, due to diversity of possibly involved disciplines in embedded systems industries. Creating an advanced embedded system can easily encompass more than a half of these typically involved engineering disciplines:

- mechanics
- electrical/electronics
- software
- control (with the previous three usually recognised as “mechatronics”)
- ergonomic and aesthetic design
- signal processing
- optical
- thermal
- acoustic
- chemical
- nuclear

What would be a behavioural description common for all these disciplines?

We focus our consideration on descriptions of a system of subsystems which interact through localised interfaces. As the phenomenon of interest we look at *localised interactions in time and space*.

The most generic concept capable to discuss the observed phenomenon operates in terms of transformations of energy. However, we find the notion of energy and any possible notation for describing it conceptually too distant from, for instance, Software Engineering practice, which is the most important integrating discipline for embedded systems.

We prefer to abstract the interaction (in energy domain usually named “transformation”) into the notion of *event*, which captures *change*, in discrete as well as continuous (i.e. hybrid) systems. We take an event as indication of change in time and space. As an abstraction of “time and space” we choose the notion of *interface*. This concept is closely related to that of component, which is our desired way to pursue the matter of systems decomposition. Finally, for describing the behaviour of events on interfaces in time, we are inclining to an operational semantics, i.e. a state transition system which describes interaction of events dependent on the state of a system.

In order to be applicable wide enough, the chosen framework must not impose any discipline-specific bindings (tools of methodologies) within the targeted domain. What defines the abstraction level at which we want to operate in the domain are the system aspects we want to tackle with our modelling paradigm. The following list gives the operational requirements we impose on this modelling paradigm – the *NeoMultiModel*:

1. Facilitating concurrent engineering and subcontracting by providing a framework for structural and behavioural specifications interchangeable among domain’s disciplines
2. Improving reuse of existing in-house as well as off-the-shelf (third-party) components, and by that substantially decrease time-to-market and increase reliability of the total system
3. Reducing testing costs via reuse of already verified components’ interfaces and allowing automated generation of test cases for different modules and their combinations
4. Supporting products line reference architecting, for boosting (controlled) product diversification
5. Facilitating release and variant management.

In order to achieve these, we propose the following technical requirements:

1. Applicability for describing interfaces of traditional components from all involved engineering disciplines
2. Verifiability of static interface specifications (components’ assembly consistency)
3. Verifiability of interface interaction protocols, including verifiability of temporal aspects of the interaction (components’ behavioural consistency)
4. Hierarchical decomposition
5. Compositional synthesis.

An encouragement in believing in feasibility of such a modelling framework, based on state transition semantics, was found in the works of Thomas Henzinger and Luca de Alfaro [13, 7, 8, 9]; this paper is basically exploring applicability of the referenced notions within embedded systems industries, with a high demand for predicting certain structural and behavioural aspects of their products. The inspiration with the mentioned works has influenced the basic choices of the proposed modelling framework – *NeoMultiModel* – with a dominating intention to define it in a way which allows applying and evaluating suitability of the referred and similar formalisms.

5. THE NOVEL MULTIDISCIPLINARY MODEL *NeoMultiModel*

This section further specifies a conceptual foundation of the NeoMultiModel modelling framework, to be implemented by a suitable formalism.

5-1 Definitions of basic notions

A *component model* is the set of component types, their interfaces, and, additionally, a specification of the allowable patterns of interaction among component types [6].

A *component* is a functional entity, functionally described only by its interaction interfaces, and possibly with other non-functional constraints.

Functional description of interfaces (i.e. interaction among components) fully defines *WHAT*-, *HOW*- and *WHEN*-aspects of interactions taking place at interfaces, i.e. their *ports* and *protocols*. *WHAT*-aspect is a sum of interactions instances that can take place on an interface. *HOW*-aspect specifies the structure of interaction instances regarding the order of interactions. *WHEN*-aspect places interaction instances in the absolute time, or relative to some events (relative occurrences of inputs and outputs). Therefore, a language for describing interaction protocols must be capable of expressing structure, order (possibly concurrency) and timing of interaction events.

An *event* captures an observable occurrence which abstracts a change in time and space. Interaction events are characterized by location, the occurrence time instance and relative temporal distance from other events; events are instant, i.e. with a zero duration time.

A *total interface* belonging to a component, is statically described by groups of input and output ports related to component's functionalities, and dynamically by interaction protocols of events on the ports. A group of inputs and outputs which allow access to a particular component's functionality is a partial – “functional” – interface. Therefore, a total interface is a collection of all functional interfaces of a component (see Figure 3).

Ports are functional interaction points of an interface (i.e. component). Ports of a component, together with their dynamic behaviour expressed in protocols, form a component's interfaces. Each port has a type: direction (is input or/and output), nature of the quantity it inputs or outputs and the value range of that quantity. Ports express *WHAT* exactly can be communicated through the interface resources.

A *protocol* is a description of order (concurrency) and preferably time distances between activations of input and output ports (events). Protocols describe *HOW* is the interaction between components (i.e. interfaces) logically/causally ordered, and for timed models it adds *WHEN* the interaction takes place. Components may produce outputs not causally related to their input activations (“generator” components).

Implementation in a broad sense are together all logical (usually in “software”) and physical (hardware from all the disciplines) realisation of an engineered system. Implementation in the software sense is all the source code resulting in executable code.

Interfaces are abstraction of behaviour of a system. They are abstract because they model high-level behaviour of the system and its components, and only to a certain depth of detail. Otherwise they would be implementation. The model, i.e. abstraction of the implementation, is necessary because an implementation is far too complex to comprehend in reasonably short time.




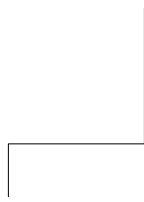


5-2 Graphical notation

There are uncountable graphical notations which represent some connected parts into a larger assembly. Among these, there are surely many suitable for our proposed component model. However, we draw our own simple symbols (Table 1), in order not to implicitly inherit any of semantics before we adopt one which addresses our issues. When NeoMultiModel becomes operational, we can evaluate existing graphical notations and consider adopting one of them.

Already at this (early) stage of its development, NeoMultiModel is open to be complemented with arbitrary discipline-specific tools and methods (see Section 6 and Figure 4), but also with languages at the system level of abstraction, as SysML or UML (when some of its numerous views happen to be handy for a specific aspect of a

system description). However, in this proposal only the following one type of the diagram is used, in order to address two issues of the popular system-level modelling formalisms: putting forward one principal view (diagram) surely understandable without any training by all possible engineering disciplines, and proposing it in such a way that formalisation would be feasible by a relatively broad spectrum of existing formal semantics.

Table 1 Graphical entities required for modelling with NeoMultiModel

	Component "PartOfTheWorld"	 opToTheWorld	Output port "ToTheWorld"
		 ipFromTheWorld	Input port "FromTheWorld"
	Connection between (compatible) ports or interfaces	 iopToOrFromTheWorld	Port "ToOrFromTheWorld" for which the orientation is not yet determined or is bidirectional
		 iAFunctionality	Interface "AFunctionality" capturing a collection of ports

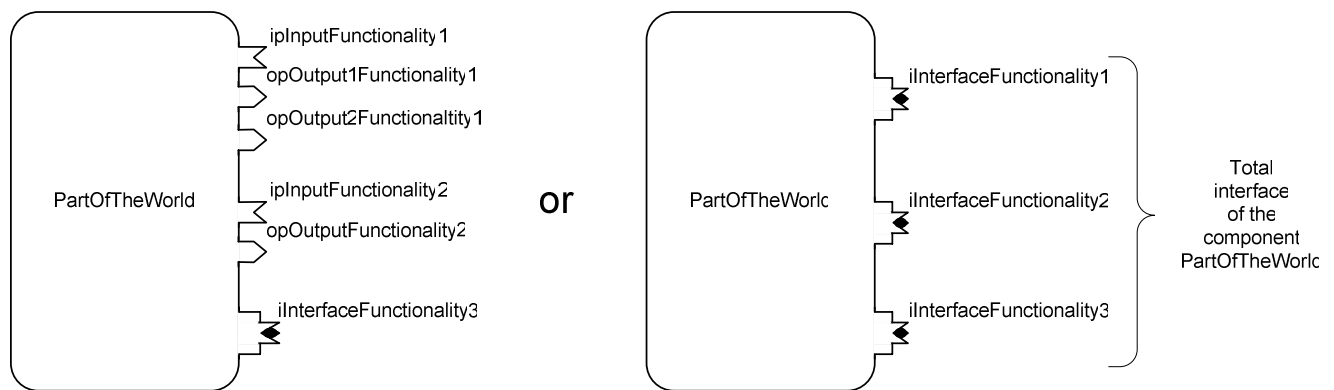


Figure 3 One example of using the presented graphical notation

5-3 Semantical formalism

As it has been already motivated, the formalism for implementing semantics of the proposed framework for multidisciplinary component model NeoMultiModel is here left open, since the point of this paper is to specify and motivate requirements for such a modelling framework. To our knowledge, and as presented in Section 7, there are several formalism which in one or another way capture the syntax and semantics of interfaces, of which already referred [13, 7, 8, 9] seemed the most close of what is specified in this paper. However, we are curious about any other approach which can fulfil our requirements, operational and technical (presented in Section 4), fit into the graphical representation understandable to engineers with backgrounds in all involved disciplines, and also allow stepwise refinement of the system model corresponding with the level of detail of the outlined Requirements Engineering, which is the subject of the next section.

6. REQUIRED MODELLING WORKFLOW

On the basis of already posed requirements, and the main motivation of this research to connect the process of Requirements Engineering with the System Architecting by product modelling, NeoMultiModel should be deployed in the way presented in Figure 4.

Figure 4 indicates stages of refinement of the model of a system which correspond to the detail of information required for RE documents. For a full understanding of the flowchart, the “module-component” convention has to be explained. Many interesting complex embedded system are modular in their nature, to be configurable or customisable. In order to address the “modularity” directly, a convention is adopted that *the system components at the top hierarchical level (in the course of decomposition) are referred to as modules*. This means that modules are no different than all other subcomponents. Needless to say, top-level components – modules – are clearly multidisciplinary.

This workflow depicts the criterion in the course of system decomposition and specification where modelling with NeoMultiModel gives way for discipline-specific modelling in specialised modelling tools. Decomposition of components need not go further of the point – named *multidisciplinary decomposition boundary* – where interfaces of components produced in the last decomposition step all belong to a single engineering discipline (for instance software or mechanical engineering). From this point on, the parent multidisciplinary component, i.e. its interface, represent specification of the monodisciplinary child components (see also Figure 7). Formal consistency verification of the system model is possible above this decomposition boundary, therefore it is important that in presence of architecture modifications, monodisciplinary models (i.e. their interfaces) are always in synch with the interfaces of NeoMultiModel at the decomposition boundary. The multidisciplinary decomposition boundary is in the workflow marked with the conditional element “Is the component monodisciplinary?”.

Further Figure 4 indicates with arrows in black dots peaces of information in the RE documents (indicated on the left) which come directly from the NeoMultiModel modelling process. Depending on the RE process definition in an organisation, Component Requirements Specification (CRS) may contain only specifications of a few top levels of the system decomposition hierarchy, or the complete decomposition. The arrowed flow upwards on the right indicates the modelling details necessary to animate the specified product architecture for presenting the behaviour of the product concept at the stakeholders level – which captures an important communication aspect of the product modelling process.

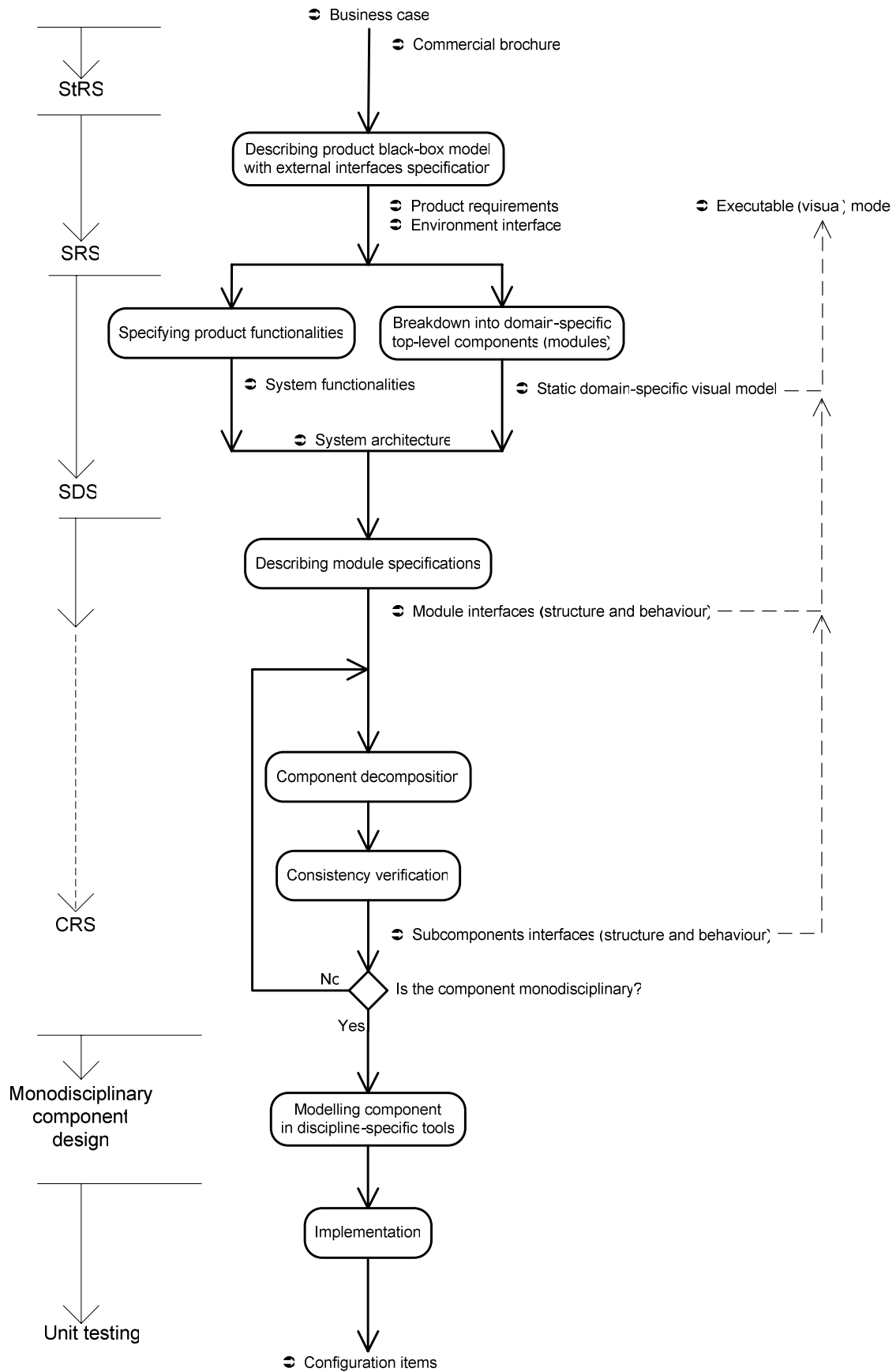


Figure 4 NeoMultiModel refinement workflow

7. WORK IN PROGRESS

A few aspects of the further definition of the specified modelling framework are subject of active research within different initiatives of the Neopost Technologies and her partners, with the intention to identify soon the best candidate formalism(s) for implementing the NeoMultiModel formal semantics.

Discipline-independent modelling notations for multidisciplinary systems is basis of the Dutch section of the hardware-software codesign- and interoperability-inspired European ITEA project TWINS [14]. Within the TWINS consortium, two groups of the Technical University of Eindhoven are proposing their suite of process algebra for modelling discrete and hybrid systems. Chi process algebra of the Systems Engineering group [12] is capable of modelling timed hybrid systems; mCRL2 algebra of the OAS group [11] is suitable for modelling discrete systems, and equipped with model checking tools. mCRL2 represents also the bridge to formally check the Chi models, since the two groups have established a translator from Chi to mCRL2. These modelling suite is being evaluated on a case study of modelling behaviour of Neopost 2- or 3-module DS-86 envelope filling system in Figure 5, of which the paper path of empty envelopes and the documents fed, folded and inserted in the envelopes is given in Figure 6.

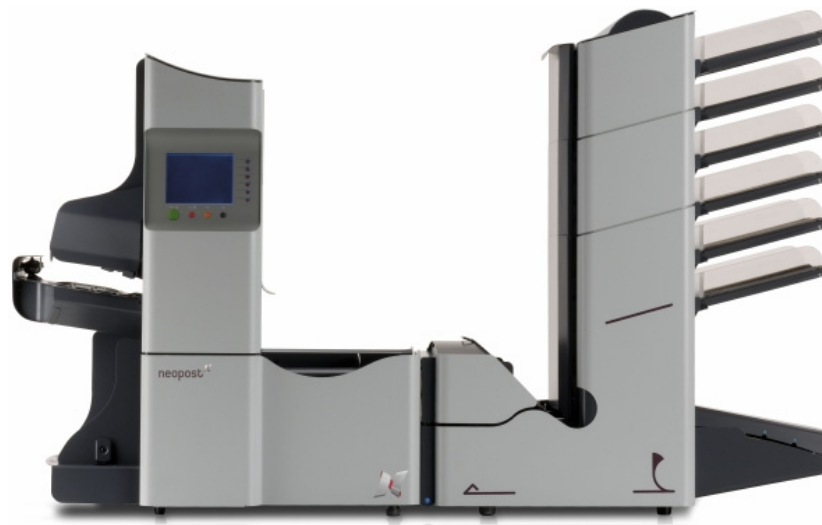


Figure 5 DS-86 envelope filling system

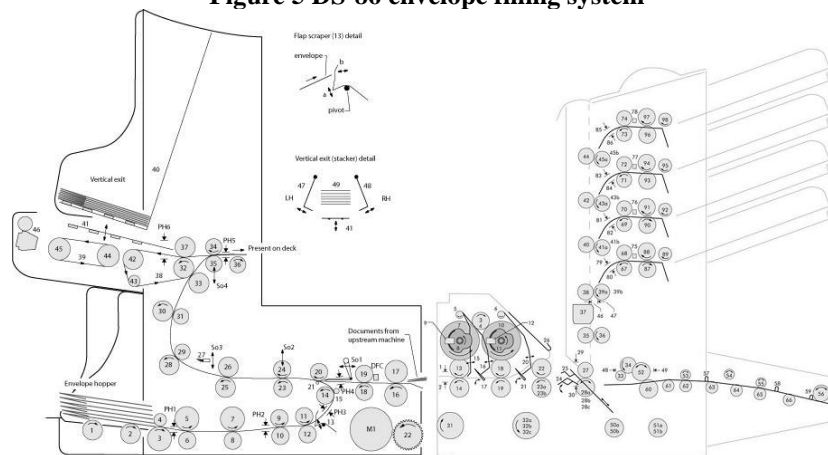


Figure 6 Paper path layout of the DS-86

In collaboration with the State University of Groningen and Technical University of Twente in form of an internship project [1], formalisation, visualisation and verification of interfaces is pursued using the timed automata-based tool UPPAAL [15]. This research initially addresses also an investigation on *substitutability* of an interface by a collection of connected interfaces, as illustrated in Figure 7.

Besides the two described research topic, there are several easily foreseeable ramifications towards further instrumentalisation of NeoMultiModel within our Systems Engineering process. One of the hot topics is for instance optimisation of energy consumption in various parts of the system for all different modes of operation.

The desired operational semantics for the NeoMultiModel would suite this topic very well, since a state-based description lends itself ideally for analysing this aspect of system's behaviour. If state descriptions for each component within a system model carry information of energy consumption for each state in which a component can be, executing such a model would easily reveal energy consumption profiles along all the working models. By attaching budgets to components – one more element from the modelling to enrich the Component Requirements Specification – a budget-driven development [10] becomes one of the features of modelling systems with NeoMultiModel.

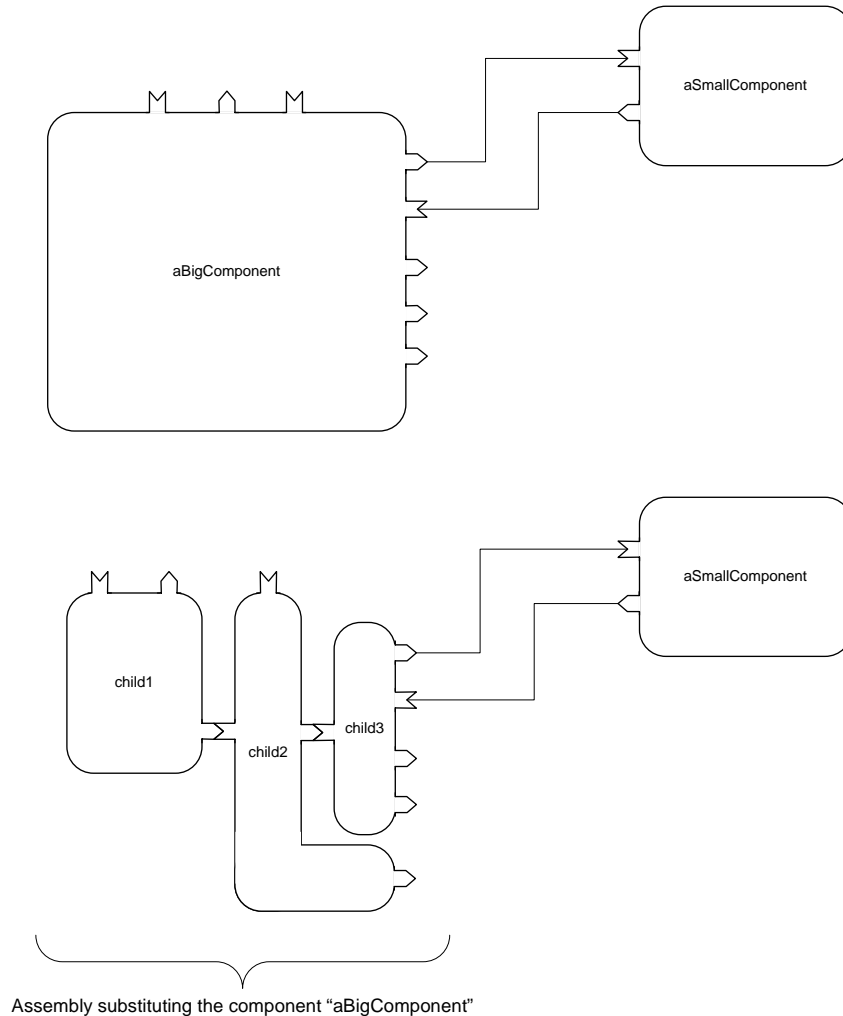


Figure 7 A substitutability case

8. CONCLUSIONS

This paper presents motivation, notional concept, initial graphical notation, modelling workflow and requirements for semantics for an interdisciplinary component model for modelling multidisciplinary embedded systems at the system level. The proposal builds on feasibility to marry in one modeling paradigm the two aspects of system architecture:

- *structure* as result of structural decomposition (with desired properties of static visualisation, composability – modularity, verifiability, managed dependencies through interfaces)
- *behaviour* as result of functional decomposition (with desired properties of behavioural visualisation – executability, configurability – modularity, verifiability, temporality)

As explained in Section 6, with only generic restrictions on describing localised interfaces with their *WHAT*-, *HOW*- and *WHEN*-aspects, this discipline-independent system-level model facilitates interoperability of any tools useful for discipline-specific refinement of monodisciplinary components. Although specified to be straight-forwardly susceptible to formalisation, the simple graphical notation stays accessible to a broad base of

engineers and stakeholders with different backgrounds and roles, also thanks to the proposed rapid behaviour animation quality. Existence of both properties of formalisation (therefore verifiability) and executability (therefore simulation and visualisation) guarantees delivery of the key enabler for sustainable project management: predictability.

Finally, the fulfilment of the starting demand of fitting this novel modelling approach into the established Requirements Engineering practices, can be illustrated on the basis of the “Devil’s triangle” of product creation management; Figure 1 suggests that the instrument for reconciling the three conflicted concerns of quality, cost and lead time is “multiparameter multidisciplinary optimisation”. Although this paper does not say anything about how to set-up and manage such an optimisation, from the elaborations in this text it can be concluded that the Requirements Engineering is the tool to take care about the “multiparameter”, and the NeoMultiModel complements it by addressing the “multidisciplinary” in such a optimisation.

The paper contributed an exhaustive set of requirements for a formalism to semantically implement the proposed modelling framework. Some reported ongoing activities indicate that the next step of realising NeoMultiModel is identification, deployment and evaluation of these and possibly other formalisms to suitably implement semantics of this modelling paradigm.

9. REFERENCES

- [1] Ben Sikkens: Interactions in Inserting Systems; Neopost Technologies B.V. internal report, 16. May 2008
- [2] Edward A. Lee: What’s Ahead for Embedded Software; IEEE Computer, September 2000, pp. 18-26.
- [3] Edward A. Lee, Haiyang Zheng, Operational Semantics of Hybrid Systems; Invited paper in Hybrid Systems: Computation and Control: 8th International Workshop, HSCC, LNCS 3414, Zurich, Switzerland, March 9-11, 2005
- [4] Grady Booch: Artifacts and Process; IEEE Software, November 2007, pp. 11-18
- [5] IEEE Standard Glossary of Software Engineering Terminology: definition of Software Quality; IEEE Press, New York, US, ISBN: 1-55937-067-X, 1990
- [6] Ivica Crnkovic, Magnus Larsson, eds: Building reliable component-based software systems; Artech House Inc, ISBN 1-58053-327-2, 2002, p.17
- [7] Luca de Alfaro and Thomas A. Henzinger, Interface automata, Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), pp. 109-120, ACM Press, 2001
- [8] Luca de Alfaro and Thomas A. Henzinger, Interface-based design, in Engineering Theories of Software-intensive Systems (M. Broy, J. Gruenbauer, D. Harel, and C.A.R. Hoare, eds.), NATO Science Series: Mathematics, Physics, and Chemistry, Vol. 195, pp. 83-104, Springer, 2005
- [9] Luca de Alfaro and Thomas A. Henzinger, Interface theories for component-based design, Proceedings of the First International Workshop on Embedded Software (EMSOFT), Lecture Notes in Computer Science 2211, pp. 148-165, Springer, 2001
- [10] Maurice Heemels, Gerrit Muller: Boderc: Model-based design of high-tech systems, Embedded Systems Institute, the Netherlands, ISBN-13: 978-90-78679-01-1, ISBN-10: 90-78679-01-8, December 2006
- [11] OAS group, Design and Analysis of System, Technical University of Eindhoven: http://www.win.tue.nl/oas/en_index.html, June 2008
- [12] Systems Engineering Group, Technical University of Eindhoven: http://w3.wtb.tue.nl/nl/organisatie/systems_engineering/systems_engineering_home/, June 2008
- [13] Thomas A. Henzinger, Slobodan Matic, An interface algebra for real-time components, Proceedings of the 12th Annual Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE Computer Society Press, pp. 253-266, 2006
- [14] TWINS-ITEA project consortium: www.twins-itea.org, June 2008
- [15] Uppsala University, Aalborg University: UPPAAL, www.uppaal.com, June 2008