A COMPARISON OF SIX LANGUAGES FOR SYSTEM LEVEL DESCRIPTION OF TELECOM APPLICATIONS

Axel Jantsch[1], Shashi Kumar[2], Ingo Sander[1], Bengt Svantesson[1], Johnny Öberg[1], Ahmed Hemani[1], Peeter Ellervee[3], Mattias O'Nils[4]

[1] Royal Institute of Technology, Stockholm, Sweden
[2] Jönköping University, Jönköping, Sweden
[3] Tallinn Technical University,Tallinn, Estonia
[4] Mid Sweden University, Sundsvall, Sweden

Abstract: *Based on a systematic evaluation method with a large number of criteria we compare six languages with respect to the suitability as a system specification and description language for telecom applications. The languages under evaluation are VHDL, C++, SDL, Haskell, Erlang, and ProGram. The evaluation method allows to give specific emphasis on particular aspects in a controlled way, which we use to make separate comparisons for pure software systems, pure hardware systems and mixed HW/SW systems.*

1 INTRODUCTION

Language evaluation and comparison is difficult because of its large number of influencing factors, many of which are difficult to quantify. The outcome of most evaluations is therefore a subjective judgement which inherits its credibility from the individuals involved. Moreover, an educated debate about this judgement is rarely conclusive because of different priorities given by different people which are often not explicitly formulated and agreed upon. Thus, an argument by person X, stating that language A is superior to language B due to smaller synthesis results, would typically be countered by person Y by emphasising, that the simulator for language B is much faster allowing higher design efficiency. Although it is very difficult to quantify these and many other issues and to agree on defined priorities, more transparency and explicitness is absolutely necessary to make progress in the discipline of language and tool evaluation.

Based on a systematic method which is described in detail elsewhere [10], we present a comparison between several languages and illustrate, how giving high or low importance to a particular aspect affects the relative performance of a language.

2 EVALUATION METHOD

2.1 Scope of the Method

The evaluation method is targeted towards system specification languages for complex telecom applications. It is based on several assumptions:

- The design process defines separate phases for specification and design and requires separate specification and design documents. Pure requirements, functional or not, are also not considered part of the specification document. Hence, if requirements are explicitly formulated we assume that this is done in a separate requirements definition document.
- It is assumed that the specification document should capture the externally visible behaviour of the system and should avoid internal design and implementation decisions as much as possible.
- It is assumed to be an advantage if the specification document is amenable to analysis and synthesis tools. In fact, we assume that the more tools and methods can work with the document the better it is.
- As noted several times the target applications are complex systems, not simple systems that can be coded directly by one person in one week.
- The application area is telecommunication. We expect that complex electronic systems in other areas, e.g. in the automotive industry, exhibit similar characteristics, but we have not analysed other areas.

In the following two subsections we elaborate some of our assumptions concerning the purpose of the specification document and the application area.

2.2 Requirements for a Specification Document

In a product development process the specification is typically the first document, where the extensive discussion of many aspects of the problem leads to a first proposal of a system which shall solve the given problem. Hence, the purpose of the specification document is twofold:

1. It is a means to study if the proposed system will indeed be a solution to the posed problem with all its functional and non-functional requirements and constraints, i.e. to make sure to make the right system.
2. It defines the functionality and the constraints of the system for the following design and implementation phases.

From these two purposes we can derive several general requirements for a specification method.

A. **To support the specification process**: To write a specification is an iterative process. This process should be supported by a technique which allows the engineer to add, modify and remove the entities of his concern without a large impact on the rest of the specification.

B. **Analysable**: The specification should be analysable in various ways, e.g. by simulation, formal verification, performance analysis, etc.

C. **High abstraction**: The modelling concepts must be at a high enough abstraction level. The system engineer should not be bothered with modelling details, which are not relevant at this stage.

D. **Implementation independent**: The system specification should not bias the design and implementation in undesirable ways. System architects must be given as much freedom as possible to evaluate different architecture and implementation alternatives. Products are frequently developed in several versions with different performance and cost characteristics. Ideally the same functional specification should be used for all versions.

E. **Base for implementation**: The specification should support a systematic technique to derive an efficient implementation. This is in direct conflict with the requirements C and D.

2.3 Application Characteristics

Systems in a particular domain exhibit many characteristics to a different extent. In fact, the difference between different domains is usually not the absence or presence of characteristics but the degree of importance of different characteristics.

Our evaluation is targeted towards digital telecommunication systems. Such systems consist of signal paths and a reactive control system. A signal path consists of dataflow functional blocks operating on streams of data with potential high data rates. The reactive control system has typically lower performance constraints, is control dominated and has sometimes large memories for configuration data and system state. The following characteristics are important for this application domain:

- *Stream processing*: The system transforms streams of data according to simple protocol transformations or complex mathematical transformations with sometimes high performance requirements.
- *Complex control:* The system can be in many different states and modes, e.g. some of them are responsible for the normal operation, some for start-up and configuration, some for testing and diagnosis, others for detecting and handling error conditions, etc.
- *Well defined timing:* Environment and requirements establish defined timing constraints which are important for the control and the stream processing part.
- *Spatial distribution:* An integrated functionality is sometimes implemented at spatially separated locations.

- *Versatile interfaces:* The system is typically connected with the environment with various standardized interfaces and protocols.
- *Large memory:* The behaviour of both control and stream processing depends on the system's state and configuration, which sometimes require large memories used in an irregular manner.

Different parts of a telecom system exhibit different characteristics ranging from pure signal processing to control dominated system management.

2.4 Evaluation Criteria

The definition of evaluation criteria is as difficult and as important as the evaluation of languages itself, because the criteria and their relative weights basically determine the outcome of the evaluation process. We base our work on the studies described by Ardis et al. [1], Narayan and Gajski [2] and Davis [3]. To a limited extent we also used the criteria discussed by Nordström and Pettersson [4]. In all these reports a set of criteria is selected based on the assumption, that if a criterion is fulfilled by a language to a high degree, the language can be more effectively used and the design process will on average result in a better product than when the criterion is not fulfilled. This excludes the design process and the designer's skill from the language evaluation. It has the disadvantage that the dependence of the end product quality on a given criterion could be misjudged.

An alternative in the selection of criteria is taken by Lewerentz and Lindner [5]. There, the main criteria are properties of the resulting model, such as liveness and correctness. Although these are the criteria of ultimate importance, they are influenced by many factors related to the design process, which had to be identified and filtered out before establishing valid conclusions about the influence of the language on these properties.

Starting with the criteria discussed in [1, 2, 3, 4], we add new criteria, divide them into four groups, namely modelling, analysis, synthesis, and usability related aspects, as illustrated in figure 1. These groups are assessed independently from each other, which means a language is subject to four different assessments rather than one. The modelling group is further divided into aspects related to computation, communication, data and time. This division is based on the observation that these four modelling aspects can be analyses separately as discussed in [9].

This list of criteria is of course to some extent arbitrary, as is the case for any similar kind of evaluation. The list is perhaps not complete and the criteria are not orthogonal and independent from each other. Not all the criteria are on the same level, some could be merged into a single criterion. Others could be refined and split into criteria covering certain aspects in more detail.

We have introduced weights for each criterion to account for overlap

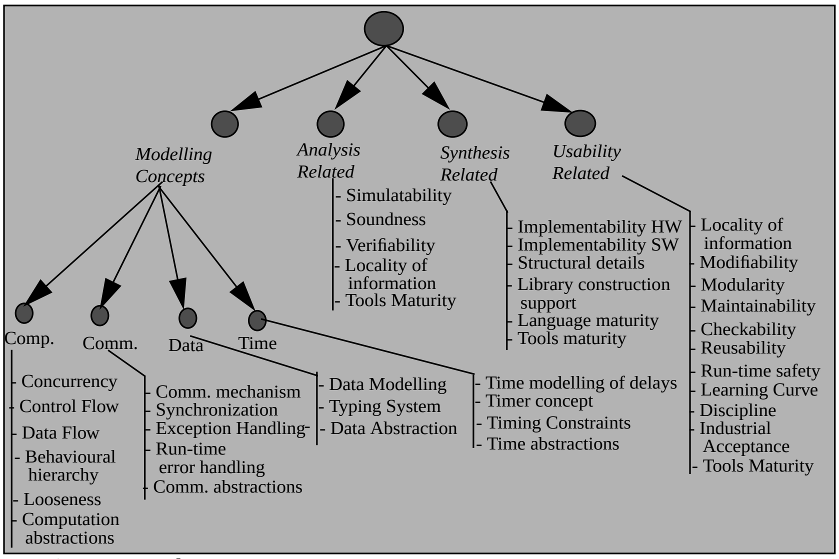

Figure 1. Evaluation criteria

between criteria and to emphasise the particular purpose of the evaluation, which is language evaluation as opposed to tool or design process evaluation. Furthermore, the criteria weights reflect the focus on specification rather than implementation. The weights allow to define priorities among the criteria but avoid implicit preferences by selecting or dropping certain criteria. The weight factors that we use in this comparison are listed in table 2. For more details on the criteria and their weights see [10].

2.5 Evaluation Mechanism

The objective of the evaluation is to get quantitative parameters to make it possible to compare the suitability of various languages for specification of systems in a given application domain. It is possible that the evaluation may conclude that language A is better than language B, language A is highly suitable for description but difficult to synthesize, or that language A is more suitable for large systems and language B is more suitable for small systems.

The method uses evaluation functions Φ, which produce a suitability index depending on the evaluated language, the application area, the system size, and the design objectives, as illustrated in figure 2. Some factors are implicit in this scheme and therefore not explicitly visible in figure 2. The design phase affects the selection of criteria, the criteria weights, the context weights and perhaps even the language weights. For a different design phase the entire evaluation must be thought over again.

C, K, and L are numerical vectors with one element for each criterion. The

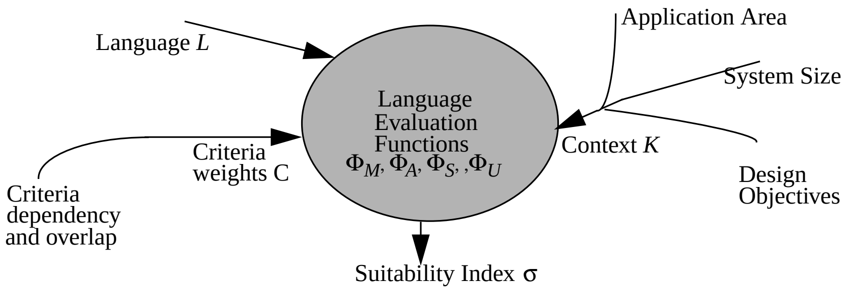

Figure 2. Suitability Vector is a four tuple

function Φ and the suitability index σ is defined by the following formulas:

$$\Phi(C, L, K) = \frac{\sum_{i=1}^{n} l_i c_i k_i}{\sum_{i=1}^{n} c_i k_i} \qquad \sigma = \langle \Phi_M, \Phi_A, \Phi_S, \Phi_U \rangle$$

By restricting the range to $K_i, C_i \in [0,1]$ and $L_i \in [-1, 1]$ the method guarantees a desirable metric as discussed in detail in [10]. To facilitate interpretation of these vectors and of the results the method uses following mapping of symbols to numbers:

K : (IRRELEVANT(IRR) ↔ 0.0, UNIMPORTANT(UNI) ↔ 0.25,
RELEVANT(REL) ↔ 0.5, IMPORTANT(IMP) ↔ 0.75, ESSENTIELL(ESS) ↔ 1.0)

L : (VERY POOR (VEP) ↔ –1.0, POOR ↔ –0.5,
FAIR ↔ 0.0, GOOD ↔ 0.5, EXCELLENT (EXC) ↔ 1.0)

Φ : (UNACCEPTABLE (UNA) ↔ [–1, –0.5), UNSUITABLE (UNS) ↔ [–0.5, 0),
SUITABLE (SUI) ↔ [0, 0.5), PROPER (PRO) ↔ [0.5, 1.0])

The purpose of these formulas is not to make the evaluation more objective but to make it more *transparent* by separating different influencing factors. The hope is that the identification and isolation of different influencing factors makes the assignment of proper weight factors easier. For instance it is easier to give a good answer to the question: “How important is the criterion of soundness for small sized control oriented applications during rapid prototyping?”, than it is to answer: “How important is the criterion of soundness for my company or my department?”. Thus, by splitting the large complex assessment into many smaller assessments the individual decisions become easier and the process of merging many small factors into one big decision is made more transparent and can be fine-tuned, rejected or accepted with confidence. It is needless to say, however, that the assignments of weights and the formula for computing the suitability index is still very subjective.

3 LANGUAGES UNDER EVALUATION

The languages under evaluation represent different paradigms. Erlang [6],

Table 1. Language assessment vectors

	L (Erlang)	*L* (C++)	L(Haskell)	L(VHDL)	L(SDL)	L(ProGram)
Structural hierarchy	GOOD	POOR	VEP	EXC	EXC	GOOD
Concurrency	EXC	VEP	FAIR	EXC	EXC	GOOD
Static processes	EXC	VEP	VEP	EXC	EXC	EXC
Dynamic processes	EXC	VEP	VEP	VEP	EXC	VEP
Control flow	EXC	EXC	EXC	GOOD	EXC	EXC
State machines	FAIR	FAIR	FAIR	GOOD	EXC	GOOD
Programming constructs	EXC	EXC	EXC	EXC	GOOD	POOR
Data flow	EXC	GOOD	EXC	EXC	POOR	POOR
Behavioural hierarchy	GOOD	GOOD	EXC	EXC	GOOD	GOOD
Looseness	GOOD	POOR	EXC	FAIR	GOOD	FAIR
Computation abstractions	GOOD	GOOD	EXC	GOOD	GOOD	VEP
Communication	FAIR	VEP	VEP	POOR	GOOD	GOOD
Synchronization	FAIR	VEP	VEP	GOOD	GOOD	GOOD
Exception handling	GOOD	FAIR	VEP	POOR	GOOD	GOOD
Run time error handling	EXC	FAIR	POOR	FAIR	FAIR	FAIR
Communication abstractions	POOR	VEP	POOR	POOR	EXC	FAIR
Data modelling	VEP	EXC	EXC	GOOD	GOOD	POOR
Typing system	VEP	FAIR	EXC	GOOD	GOOD	FAIR
Data abstractions	VEP	EXC	EXC	GOOD	GOOD	POOR
Timing modelling of delays	VEP	VEP	VEP	EXC	VEP	POOR
Timer concept	EXC	VEP	VEP	FAIR	GOOD	POOR
Timing constraints	POOR	VEP	VEP	POOR	POOR	POOR
Time abstraction	POOR	VEP	POOR	POOR	GOOD	POOR
Testability/Simulation	EXC	EXC	EXC	EXC	GOOD	POOR
Soundness	GOOD	VEP	EXC	POOR	GOOD	POOR
Verifiability	GOOD	FAIR	EXC	FAIR	GOOD	GOOD
Locality of information	GOOD	EXC	GOOD	GOOD	GOOD	POOR
Tools maturity- analysis	EXC	EXC	POOR	EXC	POOR	VEP
Implementability HW	VEP	POOR	VEP	EXC	VEP	EXC
Implementability SW	EXC	EXC	FAIR	FAIR	FAIR	GOOD
Structural details	VEP	VEP	VEP	EXC	VEP	FAIR
Library construction support	EXC	EXC	EXC	EXC	GOOD	VEP
Language maturity	GOOD	EXC	GOOD	EXC	GOOD	POOR
Tools maturity - synthesis	GOOD	EXC	POOR	EXC	FAIR	POOR
Locality of information	GOOD	EXC	GOOD	GOOD	GOOD	POOR
Modifiability	FAIR	GOOD	GOOD	GOOD	GOOD	GOOD
Modularity	GOOD	GOOD	GOOD	GOOD	GOOD	GOOD
Maintainability	GOOD	FAIR	GOOD	FAIR	FAIR	FAIR
Checkability	GOOD	POOR	GOOD	POOR	GOOD	EXC
Reusability	FAIR	GOOD	GOOD	GOOD	GOOD	GOOD
Run-time safety	GOOD	POOR	POOR	POOR	FAIR	POOR
Learning curve	GOOD	GOOD	FAIR	FAIR	GOOD	GOOD
Discipline	FAIR	FAIR	EXC	FAIR	GOOD	FAIR
Industrial acceptance	GOOD	EXC	VEP	EXC	GOOD	POOR
Tools maturity - usability	EXC	EXC	VEP	EXC	EXC	POOR

VHDL, SDL[12], and ProGram [13] have explicit concurrency; VHDL, C++, and SDL are imperative languages; Haskell [14], Erlang, and ProGram are declarative languages; C++ and SDL are object oriented languages; C++, SDL, Erlang, and Haskell have mostly been used for software development; VHDL and ProGram have been used for hardware development. One motivation for this selection was to cover different paradigms and aspects. Another practical reason was that these languages are well known by the authors.

In order to put the comparison on a solid foundation a realistic system has been modelled with all the languages, which contributes significantly to our confidence that the evaluation method is sound and the comparison is fair with respect to the given application domain.

The application example as supplied by Ericsson Telecom is an operation and maintenance system of an ATM network. ATM is an ITU-specified communication and switching technology for broadband services [7]. Operation and Maintenance (OAM) is part of ATM specifications that is responsible for detection of errors and performance degradation in the ATM network at switch level and to report it further [8]. A significant part of the OAM functionality in the ATM layer has been modeled in all languages [10]. The size of the models range from several hundred to a few thousand lines of code. However, the different OAM models cannot be compared with each other in a simple way due to several differences:

- The OAM models do not implement exactly the same functionality, even though they are very similar;
- The modelling style and the concepts used differ because the models have been developed by different persons with different objectives. This differences go far beyond what is induced by the use of different languages. For instance, the VHDL model uses bitvectors to represent ATM cells while the C++ model uses more abstract symbols; the Erlang model uses only static processes while the SDL model makes heavy use of dynamically created processes.
- The experience of the developers with the used languages varied.

For these reasons we do not attempt to compare the models in a superfluous way, like listing line numbers and development time. In a sense, the main result from the modelling activities is not the models but the analysis of language features with respect to a specific application domain. The application domain and the experience with the OAM functionality was always in the back of our minds when we analysed and discussed language concepts.

4 THE COMPARISON

We compare the languages in five different contexts. Table 1 shows the assessment vectors *L* for the languages. These are the results of the judgment

Table 2. Context vectors with different objectives

	Criteria *C*	*K*(control SW)	*K*(mixed HW/SW)	*K*(pure functional)	*K*(pure HW)	*K*(simple HW)
Structural hierarchy	0.75	UNI	UNI	IRR	UNI	UNI
Concurrency	0.75	ESS	ESS	IRR	ESS	ESS
Static processes	0.5	ESS	IMP	IRR	IMP	IMP
Dynamic processes	0.5	ESS	UNI	IRR	IRR	IRR
Control flow	0.5	ESS	ESS	ESS	ESS	ESS
State machines	0.5	IMP	IMP	IMP	ESS	ESS
Programming constructs	0.5	ESS	IMP	IMP	IMP	IRR
Data flow	1.0	ESS	ESS	ESS	ESS	ESS
Behavioural hierarchy	1.0	ESS	ESS	ESS	ESS	UNI
Looseness	0.25	REL	ESS	ESS	ESS	UNI
Computation abstractions	1.0	ESS	ESS	ESS	ESS	IMP
Communication	0.75	ESS	ESS	IRR	ESS	IMP
Synchronization	0.5	ESS	ESS	IRR	ESS	ESS
Exception handling	0.75	ESS	ESS	UNI	ESS	IMP
Run time error handling	0.5	ESS	UNI	IRR	IRR	IRR
Communication abstractions	1.0	UNI	ESS	IRR	ESS	IMP
Data modelling	1.0	IRR	ESS	ESS	ESS	IMP
Typing system	1.0	REL	ESS	ESS	IMP	UNI
Data abstractions	1.0	IRR	ESS	ESS	ESS	UNI
Timing modelling of delays	0.75	IRR	IMP	IRR	IMP	IMP
Timer concept	0.75	ESS	IMP	IRR	IRR	IRR
Timing constraints	1.0	UNI	ESS	ESS	ESS	ESS
Time abstraction	1.0	UNI	ESS	ESS	ESS	UNI
Testability/Simulation	1.0	ESS	ESS	ESS	ESS	ESS
Soundness	1.0	IMP	ESS	ESS	ESS	IMP
Verifiability	0.75	ESS	ESS	ESS	ESS	ESS
Locality of information	1.0	ESS	ESS	ESS	ESS	UNI
Tools maturity- analysis	0.25	ESS	ESS	IRR	ESS	IMP
Implementability HW	1.0	IRR	IMP	IMP	IMP	IMP
Implementability SW	1.0	ESS	IMP	IMP	IRR	IRR
Structural details	0.5	IRR	IRR	IRR	IRR	IMP
Library construction support	1.0	ESS	ESS	ESS	ESS	ESS
Language maturity	0.5	ESS	ESS	IRR	ESS	ESS
Tools maturity - synthesis	0.25	ESS	ESS	IRR	ESS	ESS
Locality of information	1.0	ESS	ESS	ESS	ESS	IMP
Modifiability	1.0	ESS	ESS	ESS	ESS	IMP
Modularity	0.25	ESS	ESS	ESS	ESS	IMP
Maintainability	0.25	ESS	ESS	ESS	ESS	IMP
Checkability	0.5	ESS	ESS	ESS	ESS	IMP
Reusability	1.0	ESS	ESS	ESS	ESS	IMP
Run-time safety	1.0	ESS	UNI	IRR	IRR	IRR
Learning curve	0.25	REL	REL	IRR	REL	IMP
Discipline	1.0	REL	ESS	IRR	ESS	REL
Industrial acceptance	0.25	ESS	ESS	IRR	ESS	IMP
Tools maturity - usability	0.25	ESS	ESS	IRR	ESS	IMP

of one or several persons for each language. In particular there were 2 persons to evaluate Erlang, 3 for C++, 2 for Haskell, 4 for VHDL, 2 for SDL and 2 for ProGram. Table 2 shows the criteria vector *C* in the second column and the context vectors K for the different objectives:

Control SW: Specification of large and complex control software as it is typical for the operation, control and management of telecom networks;

Mixed HW/SW: Specification of complex mixed HW/SW systems;

Pure functional: Specification of complex mixed HW/SW systems is similar to the “mixed HW/SW” context but with two distinguishing assumptions: (1) Explicit process level concurrency is irrelevant for the specification. One can argue that the partitioning into processes is in fact a design decision and should not be part of the specification [11]. (2) The focus is on research and factors such as tools maturity and industrial acceptance are considered to be irrelevant.

Pure HW: Specification of complex hardware systems;

Simple HW: Specification of smaller hardware systems. It is assumed that for smaller systems the need for high abstraction levels and constructs for complexity management is reduced. Thus, criteria such as behavioural hierarchy, looseness, and abstraction are deemphasized.

Tables 3 through 7 give the result of the comparison. Table 8 lists the languages which are suitable in each context, meaning, that the language performs SUITABLE or better in all four groups.

Table 3. Language comparison for large control SW

	L (Erlang)	*L* (C++)	L(Haskell)	L(VHDL)	L(SDL)	L(ProGram)
Φ_M	PRO	UNS	UNS	SUI	PRO	SUI
$\Phi_{MComputation}$	PRO	SUI	SUI	PRO	PRO	UNS
$\Phi_{MCommunication}$	SUI	UNA	UNA	UNS	SUI	SUI
Φ_{MData}	UNA	SUI	PRO	PRO	PRO	SUI
Φ_{MTime}	SUI	UNA	UNA	UNS	SUI	UNS
Φ_A	PRO	SUI	PRO	SUI	SUI	UNS
Φ_S	PRO	PRO	SUI	PRO	SUI	UNS;
Φ_U	SUI	SUI	SUI	SUI	SUI	SUI

Table 4. Language comparison for complex, mixed HW/SW

	L (Erlang)	*L* (C++)	L(Haskell)	L(VHDL)	L(SDL)	L(ProGram)
Φ_M	SUI	UNS	SUI	SUI	SUI	UNS
$\Phi_{MComputation}$	PRO	SUI	PRO	PRO	PRO	SUI
$\Phi_{MCommunication}$	SUI	UNA	UNA	UNS	PRO	SUI,
Φ_{MData}	UNA	PRO	PRO	PRO	PRO	UNS
Φ_{MTime}	UNS	UNA	UNA	UNS	UNS	UNS
Φ_A	PRO	SUI	PRO	SUI	SUI	UNS
Φ_S	SUI	PRO	SUI	PRO	SUI	UNS
Φ_U	SUI	SUI	SUI	SUI	SUI	SUI

At first it is surprising that C++ performs so poorly for control software applications. However, C++ neither supports concurrent processes nor any kind of timing, but both concepts are deemed to be important in this context. For C++ implementations typically the operating system provides these serv-

Table 5. Language comparison for complex, mixed HW/SW with low emphasis on concurrency

	L (Erlang)	*L* (C++)	L(Haskell)	L(VHDL)	L(SDL)	L(ProGram)
Φ_M	UNS	SUI	PRO	SUI	SUI	UNS
$\Phi_{MComputation}$	PRO	PRO	PRO	PRO	SUI	UNS
$\Phi_{MCommunication}$	PRO	SUI	UNA	UNS	PRO	PRO
Φ_{MData}	UNA	PRO	PRO	PRO	PRO	UNS
Φ_{MTime}	UNS	UNA	UNA	UNS	SUI	UNS
Φ_A	PRO	SUI	PRO	SUI	PRO	UNS
Φ_S	SUI	PRO	SUI	PRO	UNS	SUI
Φ_U	SUI	SUI	PRO	SUI	SUI	SUI

Table 6. Language comparison for complex pure HW systems

	L (Erlang)	*L* (C++)	L(Haskell)	L(VHDL)	L(SDL)	L(ProGram)
Φ_M	UNS	UNS	SUI	SUI	SUI	UNS
$\Phi_{MComputation}$	PRO	SUI	PRO	PRO	PRO	SUI
$\Phi_{MCommunication}$	UNS	UNA	UNA	UNS	PRO	SUI,
Φ_{MData}	UNA	PRO	PRO	PRO	PRO	UNS
Φ_{MTime}	UNA	UNA	UNA	UNS	UNS	UNS
Φ_A	PRO	SUI	PRO	SUI	SUI	UNS
Φ_S	SUI	PRO	SUI	PRO	SUI	UNS
Φ_U	SUI	SUI	SUI	SUI	PRO	SUI

Table 7. Language comparison for simple pure HW systems

	L (Erlang)	*L* (C++)	L(Haskell)	L(VHDL)	L(SDL)	L(ProGram)
Φ_M	SUI	UNS	UNS	SUI	SUI	UNS
$\Phi_{MComputation}$	PRO	SUI	SUI	PRO	PRO	SUI
$\Phi_{MCommunication}$	UNS	UNA	UNA	UNS	PRO	SUI
Φ_{MData}	UNA	PRO	PRO	PRO	PRO	UNS
Φ_{MTime}	UNA	UNA	UNA	UNS	UNA	UNS
Φ_A	PRO	SUI	PRO	SUI	SUI	UNS
Φ_S	SUI	SUI	SUI	PRO	UNS	UNS
Φ_U	SUI	SUI	SUI	SUI	PRO	SUI

ices. Hence, if one wants to evaluate C++ in combination with a particular operating system or if a particular aspect is not considered important, the vectors *L* and *K* have to be adjusted. In general the usage of particular tools or versions of a language will change the assessment, which makes apparent that it is difficult to draw general conclusions from specific evaluations. However, the evaluation method used here allows to analyse evaluation results and deviations from intuitive judgement.

Table 8. Comparison result

Context	suitable languages
Control software	Erlang, VHDL, SDL
mixed HW/SW	Erlang, Haskell, VHDL, SDL
pure functional	C++, Haskell, VHDL
pure HW	Haskell, VHDL, SDL
simple HW	Erlang, VHDL

5 CONCLUSION

We have presented a language comparison for specification of telecom systems. The main difficulty of such a task comes from the huge number of influencing factors and underlying assumptions and from the inherent subjectivity of the assessment by humans. We have not eliminated subjectivity and we cannot suggest a final conclusion but we have analysed different strengths and weaknesses of the languages and we have established causal relations between assumptions and evaluation results due to a systematic evaluation method. Each evaluation can still only be valid in a particular context and with respect to specific demands and objectives. However, we have shown a way to make an evaluation transparent and subject to detailed analysis and discussion by making all the assumptions and priorities as explicit as possible.

6 REFERENCES

[1] M. A. Ardis, J. A. Chaves, L. J. Jagadeesan, P. Mataga, C. Puchol, M. G. Staskauskas, J. Von Olnhausen, “A Framework for Evaluating Specification Methods for Reactive Systems - Experience Report”, *IEEE Transactions on Software Engineering*, June 1996.

[2] Sanjiv Narayan and Daniel D Gajski, “Features Supporting System-Level Specification in HDLs”, pp. 540 - 545, *European Design Automation Conference*, September 1993.

[3] Alan M. Davis, “A Comparison of Techniques for the Specification of External System behaviour”, *Communications of the ACM*, pp. 1098 - 1115, September 1988.

[4] A.Nordström, H.Pettersson, *An Evaluation of Graphical HDL Tools with Aspects on Design Methodology and Reusability*, Ericsson, Sweden, Report JR/M-97:1676, 1997.

[5] Claus Lewerentz and Thomas Lindner, ed., *Case Study “Production Cell”: A Comparative Study in Formal Software Development*, Forschungszentrum Informatik, Universität Karlsruhe, report no. FZI-Publication 1/94, Karlsruhe, Germany, 1994.

[6] J.Armstrong, R.Virding, M.Williams, *Concurrent Programming in Erlang*, Prentice Hall, 1993.

[7] M. De Prycker, *Asynchronous Transfer Mode solutions for broadband ISDN*, Series in Computer Communications and Networking, Ellis Horwood 1991.

[8] ITU-T Telecommunication Standardization sector of ITU Recommendation I.150, I.211, I.311, I.321, I.327, I.361, I.362, I.363, I.413, I.432, I.610.

[9] A. Jantsch, S. Kumar, A. Hemani, “The Rugby Model: A Framework for the Study of Modelling, Analysis, and Synthesis Concepts in Electronic Systems”, *Proceedings of Design Automation and Test in Europe (DATE)*, 1999.

[10] A. Jantsch, S. Kumar, I. Sander, B. Svantesson, J. Öberg, and A. Hemani, *Evaluation of Languages for Specification of Telecom Systems*, report no. TRITA-ESD-1998-04, Department of Electronics, Royal Institute of Technology, Stockholm, Sweden, 1998.

[11] A. Jantsch and I. Sander, "On the Roles of Functions and Objects in System Specification", Proceedings of the International Workshop on Hardware/Software Codesign, 2000.

[12] A. Olsen, O Færgemand, B. Møller-Pedersen, R. Reed, and J.R.W Smith, *Systems Engineering with SDL-92*, North Holland, 1995.

[13] J. Öberg, *ProGram: A Grammar-Based Method for Specification and Hardware Synthesis of Communication Protocols*, PhD thesis, Dep. of Electronics, Royal Institute of Technology, TRITA-ESD-1999-03, 1999.

[14] J. Peterson and K. Hammond, editors, *Haskell Report 1.4*, http://haskell.org/.