Basics for the SDL Metrics

Yury Chernov, ECI Telecom Ltd., Hasivim 30 , Petah Tikva ,49133, Israel,
voice: 972-3-9268307, fax: 972-3-9266870, e-mail: yury.chernov@ecitele.com

Yair Lahav, ECI Telecom Ltd., Hasivim 30 , Petah Tikva ,49133, Israel,
voice: 972-3-9266180, fax: 972-3-9266870, e-mail: yair.lahav@ecitele.com

Abstract

This paper presents initial research in building of the SDL metrics. The principle aim of the SDL metrics is to formally evaluate the quality of the software design of embedded/real-time systems. In particular, the SDL metrics is intended to enable a multi-alternative design based on the same requirements specification. The presented approach includes both the design and implementation characteristics of an analyzed system. Implementation characteristics are obtained by means of the code generated from the SDL specifications and loaded onto the target. One of the main advantages of the SDL metrics is that it can be used at the early development stages.

Keywords

Embedded/real-time systems, SDL design, software metrics, software quality, performance

1 INTRODUCTION

Increasing support by commercial tools has gained SDL significant additional functionality including code generation, simulation, and validation. These features turn the language into a powerful design technique. One of the main advantages of system modeling by SDL is that a system can be quickly built (using automatic code generation), tested and loaded into a target or a simulator. This opens new possibilities for formal analysis of design. It is now feasible to design several variants of a system based on the same functional requirements and to choose the best one. With traditional development techniques the design and analysis of alternatives is practically not reachable.

The present paper suggests a framework for the SDL metrics which provides a way to take full advantage of using SDL in a system modeling and design. The target domain for the suggested metrics is software design for real-time and embedded systems.

2 OBJECTIVES OF THE SDL METRICS

The suggested SDL metrics promotes the following principle aspects:

- *Measurement.*
  The SDL metrics allows the measurement of the quality of design and implementation. This is actually the basic aim of any software metrics.
- *Improvement.*
  The SDL metrics stimulates improvement of the quality of a project since it allows the measurement the quality after each increment of the design.
- *Fixing of drawbacks.*
  The SDL metrics makes it possible to point out specific drawbacks of an SDL model and thus to improve them.
- *Comparison.*
  The SDL metrics gives means for comparison of various SDL designs.
- *Multi-variant design.*
  The SDL metrics enables the design of several alternatives of a system. All designs satisfy the same requirements specification. The quality of each alternative can be estimated using the metrics and thus the best one can be selected.

3 PROPERTIES OF THE SDL METRICS

The suggested SDL metrics is based on industrial SDL supporting tools. It includes ideas and methods known software metrics [1, 2, 3], however it differs from them in several important aspects:

- Early usage.
  Several of the SDL metrics indices characterize implementation of the design. Thus while most of the known software metrics and software quality models are based on the measurement available only when a product is almost complete and design changes are too costly, the SDL metrics can be applied at an early development stage.
- Automated estimation.
  Many of characteristics of the SDL metrics can be obtained automatically from PR forms.
- Combination of design and implementation characteristics.
  The SDL metrics apply to both design and implementation. The implementation code is supposed to be automatically generated. Even when the final product does not include this code the simulation results serve as a good approximation of the real product.

4 A FRAMEWORK FOR THE APPLICATION OF THE SDL METRICS

The proposed framework includes the following steps of a system design:

1. Functional requirements.

Functional requirements may be presented by formal or near formal documents. They include the basic functions of the system, the basic requirements to the system and principle restrictions.

2. Design model.

A design model is a verified SDL specification of the system which is defined by the functional requirements. When the SDL metrics is used for multi-variant analysis of design, several design models are built. Each model represents one alternative.

3. Emulation model.

An emulation model includes a design model and the code generated from the design model. This code runs on a target or a simulator. In a multi-variant analysis one emulation model corresponds to one design model. An emulation model is the source and the object of the SDL metrics.

Emulation models are behavior-oriented. They include the following elements:

- Processes
- Environment
- Signals
- Scenario functions

Scenario functions serve to represent the system functionality defined in the functional requirements. Each scenario function is comprised of the following properties:

- An input signal or a set of input signals which are sent from the environment
- Processing of the received input signals
- Sending of a signal or signals to the environment or generation of an indication that the state of the system has changed as a result of the processing.

Scenario functions are used to obtain real-time characteristics of particular design. Naturally, different scenario functions may be of varying importance. Thus, typically they receive different weights.

Designation of scenario functions is an informal expert process. For the purposes of evaluating alternatives it is recommended to include a set of basic scenario functions rather than a complete set. This may lower the accuracy of the evaluation to some extent, but renders the evaluation process much more feasible.

Here are several examples of Scenario Functions of a public switching system: establishing a plain call, establishing a three-way call, execution of periodic trunk testing, execution of an audit.

5 BASIC CHARACTERISTICS AND INDICES OF THE SDL METRICS

The proposed SDL metrics include *characteristics* and *indices*. Characteristics are directly obtained from an emulation model. Indices are calculated from characteristics and other indices. Indices alone serve for design evaluation. One of the main principles which stands behind index building is to try to define them as normalized non-dimensional indices where possible. This allows more formal comparison of design alternatives.

The presented metrics is far from being complete. It is only the first step towards the formal evaluation of SDL-based project and formal analysis of design alternatives. And even indices from this minimal set are not all relevant for each implementation. It's up to the particular analyst to select the proper indices and to give them a proper interpretation and weight. One can use informal qualitative analysis, another may formalize it by building a multi-criteria folding.

Basic characteristics are presented in Table 1.

SDL Specification Characteristics	
Characteristic Name	**Notation**
Number of processes	n
Number of environment interacting processes	n^{en}
Process states	h_i
Process input signals	s^{in}_i
Process environmental input signals	s^{ie}_i
Process output signals	s^{ou}_i
Process environment output signals	s^{oe}_i
Process output points	o_i
Process self-signals and timers	s^{sf}_i
Process static connectivity	c_i
Process dynamic connectivity per user function	d_{ij}
Scenario function run-time performance	r_j
Context switching per scenario function	u_i
Code size	z

Table 1

Some remarks should be given:

- Indices i and j denote processes and scenario functions correspondingly.
- Self-signals are those signals that a process sends to itself. Self-signals include also timers, which are set and consumed or reset by the process itself. Those timers that are set in one process and consumed or reset in another process belong to the basic output and input signals.
- An output point is the point from where a process sends an output signal. One process state might include several output points. The same signal might be sent from several output points of the same process. In a similar manner states may be defined as input points.
- Process connectivity denotes number of process with which the current one interacts (sends signals to them and/or receives signals from them). Static connectivity includes all connections defined in an SDL specification. Dynamic connectivity includes only those that are used for execution of a specific scenario function.

Some indices coincides with characteristics, other are formally derived from characteristics. To distinguish indices from characteristic the upper case is used for the notation of indices. To simplify the

presentation of indices parameters of summing are omitted. In all cases, if not stated, summing by i is from 1 to N, summing by j is from 1 to M, where M denotes a number of scenario functions.

Indices are broken into five groups.

5.1 Size

Indices of this group reflect the size of a design model and its implementation in a straightforward way.

Total number of processes

$$N = n$$

Number of processes characterizes both design and implementation of a system. The design aspect comes from architectural breaking of the system into functional sub-systems (SDL blocks and sub-blocks) which are composed of processes. Processes are the lowest level of the hierarchy. From implementation view each process is a separate task that has its own priority. The management of these tasks can be carried out directly by a RTOS (tight integration) or indirectly by the SDL tool manager (light integration). In the latter case all SDL processes are represented by one RTOS task.

Total number of states

$$H = \sum h_i$$

Number of states characterizes both the size of a system and the complexity of processes. Indirectly it relates to the implementation as well, since the lesser number of states leads to more complicated the transitions between states are. Transitions are non-interruptible. Thus the number of states influence the context switching.

Total number of involved signals

$$S = \sum (s^{in}_i + s^{ou}_i)$$

Total number of signals includes incoming and outgoing signals to/from the environment of the system and inner signals among processes. Number of the environmental signals is the same in all design models. Number of inner signals changes from one design model to another.
The total number of signals itself is not a very informative index since signals may have parameters. In many cases a designer may define additional signals or additional parameters for a restricted set of signals to provide the same functionality. However, should the same principles be followed in signals definition in all design models, the total number of signals may become an important index.

Code size

$$Z = z$$

Actually, the code size index can be presented by two characteristics. The first is the well-known software metric indicators - LOC (lines of code) or KLOC (thousands of lines of code). However, one of the principle assumptions of our framework is that an automatic code generation is being applied. This makes the LOC index less relevant. The second a more important one is the load size in Kbytes. It is of especial importance for embedded systems when the available memory is restricted.

5.2 Environment interaction

These indices basically reflect the level of de-coupling of interactions with the environment. A “good” design implies high level of de-coupling with specific processes destined for sending and reception of environmental signals. However, this requires some redundancy. Environmental indices may be represented separately for incoming and outgoing signals.

Environment incoming signal factor

$$E^{in} = (\sum s^{ie}_i / s^{in}_i) / N$$

Environment outgoing signal factor

$$E^{ou} = (\sum s^{oe}_i / s^{ou}_i) / N$$

Environment signal factor

$$E = (\sum (s^{ie}_i + s^{oe}_i) / (s^{in}_i + s^{ou}_i)) / N$$

The environment signal factor characterizes the homogeneity of environmental signals allocation to processes. The larger the index is, the more homogeneously environment signals are allocated and, hence, the level of de-coupling is lower. However the particular value of the factor depends a lot upon the homogeneity of the allocation of all signals, not only environmental. Should this allocation be absolutely homogeneous the value of the environment signal factor will be the same for each allocation of the environment signals. So this index is more informative when all signals are allocated non-homogeneously over the processes.

The environment signal factor is less sensitive to the number of processes involved in the interaction with the environment.

Lets try to formally analyze the environmental signal factor. Lets denote the total number of environmental signals by S^e . When all signals are allocated uniformly E gets it maximal value - E^{max} . To evaluate this value we should assume that each process has S/N signals. So,

$$E^{max} = (\sum (s^{ie}_i + s^{oe}_i) /(S/N)) / N = S^e /S$$

To evaluate E^{min} we assume that environmental signals are allocated in the most non-uniform manner - all environmental signal relates to one process. Then all components with the exception of one, in the E-expression, are zeros. There can be two possible alternative assumptions about inner signals. The first is that they are distributed uniformly along processes - E_1^{min}, thus each process has (S-E)/N inner signals.

$$E_1^{min} = (S^e / (S^e + (S - S^e)/N)) / N = S^e / (N * S^e - S^e + S) =$$
$$1 / (N - 1 + S/ S^e) = 1 / (N - 1 + 1/ E^{max}) = E^{max} / (1 + E^{max} *(N - 1))$$

The second alternative assumption is a stronger one. We assume that inner signals are allocated in proportion to environmental signals. That is, all processes have one signal and the environmental process have all the remaining signals. Lets denote it E_2^{min}.

$$E_2^{min} = (S^e / (S - (N -1))) / N$$

If we assume that S>>N, that is very realistic, then we have

$$E_2^{min} = E^{max} / N$$

$E_2^{min} \le E_1^{min}$, and $E_2^{min} = E_1^{min}$ when N=1 or $E^{max} = 1$.

Number of processes interacting with the environment

$$K = n^{en}$$

Environment process factor

$$F = K/N$$

The environment process factor is sensitive to the number of processes that interact with the environment and non-sensitive to the allocation of environmental signals over these processes.

Incoming/outgoing de-coupling factor

This index (G) should characterize an average level of de-coupling between environmental incoming and outgoing signals. The basic requirements are as follows:

G = 1 when de-coupling is maximal;

G = 0 when environmental incoming and outgoing signals are allocated uniformly among processes.

Lets define incoming/outgoing de-coupling factor for the ξ-th process g_ξ. Here summing is over environmental process only, that is from 1 to K.

$$g_\xi = (| s^{ie}_i / (\sum s^{ie}_i / K) - s^{oe}_i / (\sum s^{oe}_i / K) |) \ / \ (s^{ie}_i / (\sum s^{ie}_i / K) + s^{oe}_i / (\sum s^{oe}_i / K))$$

$\sum s^{ie}_i / K$ and $\sum s^{oe}_i / K$ - average number of environmental incoming signals and outgoing signals.
After an elementary transformation

$$g_\xi = (| s^{ie}_i \sum s^{oe}_i - s^{oe}_i \sum s^{ie}_i |) \ / \ (s^{ie}_i \sum s^{oe}_i + s^{oe}_i \sum s^{ie}_i)$$

and $G = (\sum g_\xi) / K.$

5.3 Complexity

These indices characterize the complexity of processes and hence the future efforts for maintenance, debugging, adding new features and improving existing ones at the process level. The number of states index (H), which also characterizes complexity , was discussed above.

Incoming signal factor

$$L^{in} = (\sum s^{in}_i / h_i) / N$$

Outgoing signal factor

$$L^{ou} = (\sum s^{ou}_i / h_i) / N$$

Signal factor

$$L = (\sum (s^{in}_i + s^{ou}_i) / h_i) / N$$

Signal output point factor

$$Q = (\sum s^{ou}_i / o_i) / N$$

L^{in}, L^{ou}, L and Q grow when the process complexity increases.

State output point factor

$$P = (\sum o_i / h_i) / N$$

5.4 Connectivity

Indices of this group characterize connectivity among processes. Connectivity indicates two principle aspects. The first is how many processes are affected when a new feature is added or an existing one is improved. The second indicates the complexity of implementation of a new feature.

Static connectivity

$$C^{ab} = \sum c_i \ / N$$

Static connectivity factor

$$C = (\sum c_i \ / (N-1)) / N$$

Dynamic connectivity

$$D^{ab} = \sum((\sum w_j * d_{ij})/ \sum w_j) / N$$

Here M denotes the number of defined scenario functions and w_j - weight of the j-th user function given by the analytic staff.

Dynamic connectivity factor

$$D = \sum((\sum w_j * d_{ij} /(N-1)) / \sum w_j) / N$$

Process involvement index

$$V = \sum((\sum w_j * [d_{ij}])/ \sum w_j) / N$$

Here $[d_{ij}] = 1$, when $d_{ij}>0$, otherwise $[d_{ij}] = 0$.

Context switching index

$$U = \sum w_j * u_j \ / \sum w_j$$

The context switching index is always not less than the connectivity index, since during one execution of a scenario function various processes might transfer control (send message) to one another several times.

The higher the values of the connectivity indices, the more complex the functional inter-process dependence.

5.5 Performance

This group relates to the run-time performance of scenario functions.

Run-time performance index

$$R = \sum w_j * r_j \ / \sum w_j$$

Part of SDL metrics indices can be both static and dynamic. Static indices can be derived from design models and they do not depend on scenario functions. Dynamic indices relate to scenario functions. They are estimated as a weighed combination of indices per each scenario function (like run-time performance index).

The SDL metrics can be separated from the software quality problem. Many concepts and models of software measurement and software quality have been introduced through the years [2, 4]. They define different attributes, software metrics, software quality indicators and measurement models and introduce different techniques to produce software, which is better with respect to their concepts. The aim of SDL modeling in this aspect is to introduce criteria and software metrics which better suit embedded software design. Since most existing approaches have their pros and cons and explicitly or implicitly are aimed at specific domains, the most rational way is to take the existing standard [5] for criteria and to map the SDL metric indices onto it. Below is the matrix (Table 2), which describes the mapping of SDL metrics onto the software quality criteria.

Quality Criteria	**SDL Metrics Indices**															
	N	H	S	Z	E	K	F	G	L	Q	P	C	D	V	U	R
Functionality																
Reliability					x	x	x	x						x		
Usability	x	x	x						x	x	x	x			x	
Efficiency				x											x	x
Maintainability	x	x	x					x	x	x	x	x	x	x	x	
Portability					x	x	x									

Table 2

Functionality defines a set of functions and their specified properties to satisfy stated or implied needs. None of available metric indices reflect functionality of the system, because all analyzed implementation models are supposed to provide complete required functionality.

6 CASE STUDY

A case study was built to illustrate the described SDL metrics. This case study is a simplified version of an industrial project. The product is a protocol converter card. This card is an interface between a public switch and PABX. Its purpose is to convert from the standard line signaling (4-bit ABCD signaling) on the switch side to the proprietary signaling of a PABX (2-bit signaling) and vice versa. The protocol converter converts the signaling only. Tones and digits are transparent to it. Inter-talking is supported by the PABX without outgoing to the switch. The protocol converter is connected to the control device (PC) through a special port. It receives maintenance commands from the control device (set converter state, perform test). On each converter state and channel state a log report is sent to the control device.
Periodic testing of both connections (to the switch and to the PABX) is performed.

Five scenario functions were defined for real-time performance evaluation. These functions and their weights are presented in Table 3.

No	**Scenario Function**	**Weight**
1	Set protocol converter inner state	0.50
2	Perform manual testing	0.75
3	Perform periodic testing	0.75
4	Establish an ordinary call (switch side - originator)	1.0
5	Establish an ordinary call (PABX side - originator)	1.0

Table 3

Two alternatives were built.
The first is based on a “good” architecture principle. The main functions are de-coupled among blocks. Separate blocks for testing, man-machine interface (interactions with a craftsperson through the control device), protocol converter maintenance are defined. These blocks interact with each other by means of specially defined signals. The second design alternative represents “spaghetti” like principle. One block is defined. All functions with the exception of call process support (maintenance, man-machine

interface, testing), are performed by one process. Thus, number of signals and interactions among processes is decreased. However processes are more complicated.

The described SDL metric indices for both design alternatives are presented in Table 4.

Indices			Alternatives	
No	**Name**	**Not.**	**I**	**II**
1	Total number of processes	N	8	4
2	Total number of states	H	23	18
3	Total number of signals	S	141	88
4	Code size (Kbytes)	Z	1057	974
5	Environment incoming signal factor	E^{in}	0.11	0.18
6	Environment outgoing signal factor	E^{ou}	0.28	0.53
7	Environment signal factor	E	0.09	0.15
8	Processes interacting with the environment	K	3	4
9	Environment process factor	F	0.38	1.0
10	Incoming/outgoing de-coupling factor	G	0.41	0.36
11	Incoming signal factor	L^{in}	5.27	5.46
12	Outgoing signal factor	L^{ou}	5.54	6.04
13	Signal factor	L	10.81	11.50
14	Signal output point factor	Q	0.81	0.67
15	State output point factor	P	6.98	8.73
16	Static connectivity factor	C	0.33	0.84
17	Dynamic connectivity factor	D	0.16	0.37
18	Process involvement index	V	5.43	3.25
19	Context switching index	U	17.75	11.88
20	Run-time performance index (normalized)	R	1.0	0.93

Table 4

A comprehensive analysis of this example is beyond the scope of the current paper. However, we can conclude that indices that characterize environment interaction and connectivity carry the main difference between two design models.

7 CONCLUSION

The presented metrics provides means for both the evaluation of the quality of an SDL model and the comparative analysis of alternative variants of design for real-time and embedded systems. However, the suggested SDL metrics is far from being complete. It presents only a first step towards the formal estimation of SDL-based design and formal analysis of design alternatives. The main issues which require additional research refer to non-covered SDL properties (for instance, dynamic process creating) and relevant interpretation of estimations, which can be resolved empirically.

8 REFERENCES

[1] N.E. Fenton, S.L. Pfleeger : “Software Metrics, A rigorous and Practical Approach”, PWS, Boston, 1997

[2] S. H. Kan: “Metrics and Models in Software Quality Engineering”, Addison Wesley, 1995.

[3] H. Zuse: “Software Complexity: Measures and Methods”, de Gruyter, 1991.

[4] G.R Domney: “A Model for Software Product Quality”, IEEE Transactions on Software Engineering, vol.21, no.2, pp.146-162, 1995.

[5] ISO/IEC Standard ISO-9126: Software Product Evaluation - Quality Characteristics and Guidelines for Their Use, 1991.