SDL: meeting the IoT challenge

Edel Sherratt

Department of Computer Science, Aberystwyth University
eds@aber.ac.uk
http://users.aber.ac.uk/eds

Abstract. SDL 2010 offers excellent support for modelling, simulating and testing systems of communicating agents. However, it is not perfectly adapted to meeting the specific challenges presented by the Internet of Things (IoT). Three areas that pose a challenge are considered, and language adaptations that aim to address the specific needs of IoT systems developers are explored.
The first challenge concerns signal delay or signal loss on crowded networks. Signals in SDL 2010 are by default delayed by an indeterminate duration, but a facility to model delays that depend on network traffic would be desirable. A modification is proposed to enable this.
The second concerns undesirable interactions with external IoT systems. SDL 2010 supports modelling of a system within an environment populated by multiple agents. It also allows modelling of multiple interacting subsystems. However, it would be useful to be able to model interactions with external agents in a way that supported identification of threats to reliability, privacy and security of an IoT system. An adaptation of channel substructures, a construct that was dropped from SDL 96, is proposed to facilitate this.
The third and final challenge concerns the signal handling by multiple recipients. Different approaches to supporting this are considered with a view to further investigation to determine their desirability.

Keywords: SDL (Z.100), Internet of Things (IoT), modelling, simulation

1 Introduction

The Internet of Things (IoT) is made up of systems that affect many different aspects of life. IoT systems include smart homes and cities, domestic appliances, children's toys, field robotics and more [1]. All these systems collect sensor data. Some transmit the data to repositories for offline processing to support activities such as decision making or scientific investigation. Others react to the data they receive by generating signals that control physical systems such as lighting or heating in buildings, road traffic signals or physical locks or barriers. All have requirements for safety, security and reliability, and appropriate engineering processes should be followed when they are developed and deployed.

SDL 2010, a member of the ITU Z.100 family of standards [2], is a non-proprietary formalism for modelling, simulating, implementing and deploying systems that involve parallel activities and communication. With its strong theoretical basis, its excellent tool support and its established track record in distributed systems development, SDL 2010 is ideally placed to provide support for creating critical elements of the Internet of Things (IoT). SDL's capacity for modelling and simulating communicating systems that interact with the physical world has been demonstrated by numerous examples, including a railway crossing [3], a toffee vending machine [4], and automotive and building control systems [5].

More recent work [1] explored the many benefits of SDL for developing and deploying IoT systems, but also identified specific technical areas where SDL could be more closely aligned with the needs of IoT developers. Challenges include

- loss or delay of signals depending on load on a communication channel;
- unwanted interaction between a new IoT system and environmental agents.

These both stem from the fact that IoT systems depend on a shared, publicly accessible communications infrastructure.

A related concern is that signals in an SDL model are consumed by a single recipient, but addressing requirements for reliability and for privacy in an IoT system entails taking account of the fact that signals might be consumed by multiple environmental agents as well as by their intended recipient.

It should be emphasised that all these situations can already be modelled using SDL 2010, but that modelling and simulation could be made more direct and intuitive for the IoT developer by introducing the changes discussed below.

Each of the following sections addresses one of these challenges. Section 2 addresses signal delay in a busy network. It outlines a smart city scenario, identifies support provided by SDL 2010, and proposes a modification to the SDL delaying channel. Section 3 explores the problem of unwanted interactions with external systems. It introduces a simple situation involving two scientific investigations in a remote location, explores the differences between classical telecommunications feature interaction and the kinds of unwanted interaction likely to occur in the IoT, and proposes a language extension based on the channel substructure of SDL 96. Section 4 considers situations where multiple recipients respond to an event. For each of these challenges, the aim is to provide language facilites to support the work of the IoT developer while retaining the benefits of SDL. Other important IoT challenges that were also identified in [1], such as power management, co-design of hardware and software, and targeting of new IoT platforms, that relate to use of SDL and to tool support for SDL rather than to the language itself, are not addressed here but will be the subject of future work.

2 Signals in a busy network

> "The impact of deployment of an IoT system in contexts with numerous of other systems making use of the same communication resources causing potential delays and loss of messages is supported in SDL only

to a limited degree. SDL may be extended with features to specify possible loss and delay of signals based on the load on a communication path." [1].

In a busy network, delay or even loss of signals is likely. An engineer uses modelling and simulation to predict the behaviour of an IoT system in a network whose communications resources are shared with many other systems.

2.1 Scenario: Bristol Is Open

Bristol Is Open, a joint venture between the University of Bristol and Bristol City Council [1], collects sensor data from waste bins, street lights, volunteers' smart phones and GPS devices, and from many other sources. This data can be used for research or for creative activity, but a key intention is to create an *open programmable city.* Ideas include automatic diversion of traffic in response to road congestion, or text messages to notify people of problems with air quality in particular areas.

With so much data being collected from so many sources, and so many actuating signals being sent in response to that data, delay or loss of signals is highly likely. This is a real-time problem. Significant delays in notifying citizens about problems with air quality, or in establishing road traffic diversions are not acceptable, and a good model allows such delays to be predicted.

Real-time behaviour is modelled in SDL in terms of a global time that increases as a simulation proceeds. The current value of global time is accessed by means of the SDL **now** expression, and is used in conjunction with SDL timers. This model of time is not expressive enough to meet all the needs of real-time systems developers [6], and over the years modelling and simulation of real-time activities in SDL have attracted considerable attention. An intermediate representation for SDL enabled investigation of alternative meanings of time in SDL [7] as well as facilitating different kinds of model analysis by supporting tool-set integration. Real-time extensions to SDL introduced in [6] were further developed in [8]. Such approaches provide powerful mechanisms for modelling and simulating passage of time. However, providing controllable time would require a significant change to the SDL semantics [9].

2.2 Modelling delay with delaying channels in SDL 2010

Using SDL 2010 without modification or extension, the kind of delay that might be expected in a busy network can be modelled using a delaying channel, as illustrated in Figure 1a. A signal placed on a delaying channel is delayed for an indeterminate period of time, and delaying channels are used to simulate the behaviour of IoT systems communicating over a busy, shared network.

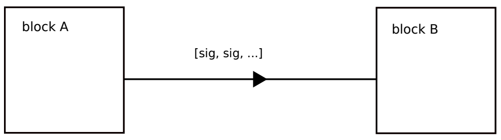
However, in order to model the behaviour of systems that are deployed in a network that also supports large and varying populations of external agents,

[1] `http://www.bristolisopen.com/`

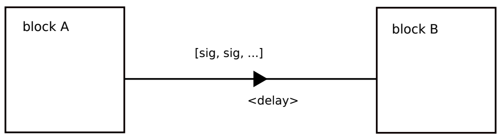
it would be useful to be able to model delays whose variability was conditioned by network traffic density. That is, engineers need to be able to design and run simulation experiments that reveal the behaviour of a system under different network traffic conditions. In other words, they need to run experiments in which network traffic acts as the independent variable controlled by the engineer, and signal loss or delay depends on network traffic.

2.3 Proposed modification

A modification to SDL delaying channels is proposed that allows the engineer to specify a delay function that takes account of traffic on the channel. This modification provides some control over how the passage of time is modelled but does not change the underlying SDL time semantics. Figure 1b) illustrates a proposed modification to SDL2010 that provides a reference to a named delaying function.



(a) Signals placed on a delaying channel are delayed for an indeterminate duration



(b) Signals are delayed by a duration that depends on channel traffic

Fig. 1: Simulating a busy network with a delaying channel

2.4 Implementing the proposed change

The textual representation (SDL-PR) for a channel is defined in Z.106 [2] using the following grammar rule

```
<channel definition> ::=
    channel [<channel name> [<encoding rules>]]
        [nodelay]
        <channel path> [<channel path>]
    endchannel [<channel name>] <end>
```

In the following version of the rule, an optional <specified delay> has been added as an alternative to **nodelay** in the rule specifying a channel definition. The nonterminal <specified delay> names a predefined function that takes an integer as input and returns a value of type duration. The function is evaluated at run time, delaying each signal by a duration that depends on the rest of the traffic on the channel.

```
<channel definition> ::=
    channel [<channel name> [<encoding rules>]]
        [nodelay|<specified delay>]
        <channel path> [<channel path>]
    endchannel [<channel name>] <end>
```

The specified delay could be implemented as a predefined runtime library from which the engineer could name the required delay. A collection of named probability distribution functions, or cumulative distribution functions would be useful for this purpose. Each distribution function could take as input the length of the delaying queue, or some combination of the delays already applying to signals in the queue, and delay the next signal accordingly.

An alternative approach, that would have allowed the developer to specify the delay was considered and rejected. The alternative approach was to define <specified delay> as a reference to an SDL procedure that could be defined by the developer, rather than as a reference to a predefined function. Although this approach would have provided more expressive power to the IoT developer, it was rejected because it would have demanded a fundamental change to SDL. A procedure has to be evaluated in the context of an SDL agent, that is, a block or process, and not a channel. Introducing a channel agent, analogous to block or process agents, would be a significant change to the current SDL semantics. For this reason, the previous suggestion – a predefined set of delay functions – is preferred.

3 Unwanted interaction with external agents

IoT systems are deployed in an environment that exposes them to external agents at every system level. SDL 2010 supports modelling of a system in an environment populated by multiple agents. It also allows modelling of multiple interacting subsystems. This section explores the extent to which SDL enables the engineer to model interactions with external agents in a way that supports identification of threats to reliability, privacy and security of an IoT system.

3.1 Scenario: scientific investigation in a remote location

Two different investigations were conducted in a remote location[2]. Each involves data collection using a wireless sensor network.

[2] This scenario is based on recent, as yet unpublished, work.

Browsing patterns of sheep The purpose of the first investigation was to discover the browsing patterns of sheep. Each sheep was provided with an ear tag that broadcast a signal at regular 15 minute intervals. The broadcast signals were picked up by devices mounted on poles, and the position of each sheep was calculated by triangulation. The positions were then forwarded to outlying stations where data was collected for further processing.

For 100 sheep, collisions were infrequent, which meant that power usage was acceptable and useful data was collected. But once the number of sheep or the reporting frequency was increased, collisions were frequent, demands on batteries became unacceptably high and data was less useful.

Environmental monitoring A number of sensors were placed in a remote location to collect data about temperature, humidity, wind speed and other environmental factors [10]. The data collected by the sensors was forwarded for central processing.

This second investigation had the potential to interfere with the first, leading to frequent collisions, re-sends, excessive power consumption and possibly unusable data.

As it happens, although the two investigations were carried out on behalf of two different organizations, they were in fact conducted by the same investigators, who ensured that the signals broadcast by the two experiments did not interfere with each other. But if the contracts had been awarded to different investigators, contention would have been likely.

3.2 Interaction in the IoT

The scenario described above is a simple one, but despite its simplicity, it provides scope for unwanted interaction between different communicating systems. The previous scenario, Bristol is Open, is complex, and provides even more opportunity for interference between different systems. The current IoT is populated by all manner of systems built using low-cost micro-controllers, single board computers and other components. Many of these devices and systems are created as a learning exercise, or as a proof of concept. Some are likely to be poorly designed, and some may even be malicious [1]. An engineer designing a system with a view to deployment in the Internet of Things must take account of such external systems, and of their implications for reliability and security of the new system. This means taking account not only of requirements for interaction within the new system, but also addressing the problem of unwanted interactions with other devices and systems in the IoT

Unwanted interaction between a new IoT system and other systems deployed on the same communications infrastructure is closely related to the well-researched problem known as *feature interaction* in the world of telecommunications services. Feature interaction occurs when new software added to an existing telecommunication system interacts in an undesirable way with the existing software. That is, new features interact with existing features, and the outcome is not what the developers intended.

3.3 Modelling and simulating feature interaction with SDL

SDL has an established record in feature interaction detection [11, 12]. When the CRESS notation was developed to communicate feature interaction problems with non-specialists, translation to SDL enabled practical simulation and exploration of those problems [12]. As Internet telephony became more prominent, SIP (Session Initiation Protocol) services and their feature interactions were modelled using SDL with the aim of detecting and prevent unwanted interactions [13].

In these examples, SDL provides a way to model an existing system and a new feature, and, with the help of tools, to simulate the behaviour of the whole system, with its old and new elements, and to discover problems such as livelock, deadlock, and interactions that reflect conflicts between the underlying requirements or assumptions for the new feature and the rest of the system.

3.4 Differences and Challenges

However, there are important differences between the conventional feature interaction problems in telecommunications and the interaction problems faced by a system deployed in the IoT, and these differences present challenges to the engineer who uses SDL to detect and prevent interaction problems.

The first concerns the resolution of unwanted interactions. In a typical feature interaction investigation, the engineer aims to detect and prevent undesirable interactions involving new and existing features, but includes both sets of features in the final system. An engineer concerned with unwanted interactions between a new IoT system and the external systems with which it will share a communications infrastructure wants to predict and prevent those interactions, but in a way that as far as possible makes the external systems invisible to a user of the new system. Ideally, the new IoT system would not have to share infrastructure; in practice, the new system must share, but should be insulated against interference.

The second difference concerns access to the communications infrastructure. In a telecommunications system, certain communications cannot be intercepted without physical intervention. In SDL terms, internal channels cannot be accessed by environmental agents. In the IoT, communications are by way of the public Internet, and can be intercepted by systems other than the one to which they belong. Moreover, the source of an externally generated signal may appear to be other than it actually is.

SDL presents a problem to the engineer who needs to explore potentially threatening interactions between a new IoT system and other systems that share the same communications infrastructure. SDL guides development towards systems that to minimize the likelihood of unwanted interactions. Channels can only transfer named signals. Internal channels are not accessible to the wider environment. Communication between a system and its environment is by way of a well defined interface across the system boundary. Interactions with the system environment are modelled as signals that are passed across the system boundary via a well defined interfacece, and environmental agents are typically only of interest insofar as different system features may demand conflicting responses to environmental stimuli. But once the new system is deployed on the public Internet, its internal communications become vulnerable to interference that was not possible in the original SDL model.

A further difference between conventional feature interaction and unwanted interaction between a new system and other, external IoT systems is that features are not created specifically to damage other features, whereas malicious systems are likely to be found in the IoT. Such systems specifically aim to intercept signals, or to create false signals, or to impersonate agents of the system under development, and modelling such systems with a view to preventing their success poses an additional challenge to the developer of new IoT systems.

A fourth and final difference between conventional feature interaction problems and unwanted interaction between a new system and its environment concerns visibility of the internal behaviour of interacting sub-systems. When interactions between features of a telecommunications system are investigated, the engineer (investigator) has access to the internal behaviour of both the existing system and the new features, and can used that access to detect and prevent unwanted interactions between new and previously existing features. When investigating differences between a new IoT system and its environment, the engineer does not have access to the internal structure of environmental agents, and must instead consider the kind of interface a potentially interfering agent might have.

3.5 Using SDL 2010 to model unwanted interaction

From the previous discussion, the following requirements can be identified:

- an ability to detect and resolve interactions in which signals from the environment affect the behaviour of new IoT system;
- an ability to model environmental interference that affects internal channels within a new IoT system, and, having ensured that the new system is resilient in the face of such interference, to remove models of external systems before implementing and deploying the new system;
- an ability to model malicious agents that deliberately subvert the proposed new IoT system, possibly impersonating parts of the new IoT system, or parts of its intended environment;
- an ability to detect of undesirable interactions without knowledge of the internal structure of environmental agents.

Key to meeting these needs is an ability to model interception of signals on internal channels by environmental agents.

With SDL 2010, undesirable transmission of signals between the environment and the system can be modelled by directly adding such unwanted transmission to the model of a system that is being developed. However, this entails modifying internal system agents and adding channel definitions to signals that represent the unwanted transmission. Even though use of signal lists makes this a reasonably clean procedure, creating the modification in order to investigate unwanted interaction, and later reversing the modification in order to generate the required system, is a source of error and vulnerability.

Similarly, it is possible to model signal interception, or injection of unwanted signals into a channel by adding a rogue block to an SDL model of new IoT system, as illustrated in Figure 2. Again, this intervention clutters the model and its later removal is another likely source of error.

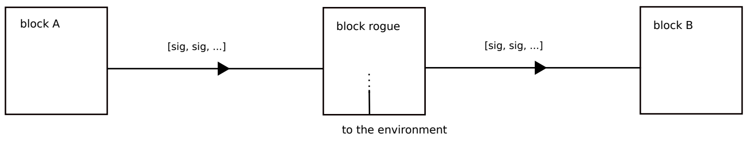


Fig. 2: A rogue block interferes with a channel in SDL 2010

3.6 Proposed Solution

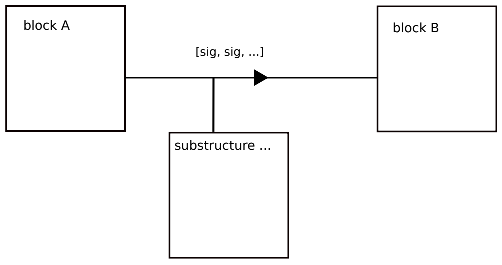
A better solution is to reinstate something very like the channel substructure of SDL 96 [4]. Channel substructures enabled the user of SDL to specify the behaviour of a channel explicitly. A channel substructure specification was like a block specfication except that it connected to external blocks, or the environment, rather than to channels. Channel substructures were dropped from SDL 2000 [14] because they were never used in practical models.

The difference between what is proposed here and the SDL 96 channel substructure is that, instead of aiming to specify the behaviour of a channel, the intention is to provide a clearly identifiable model of environmental interference with the channel.

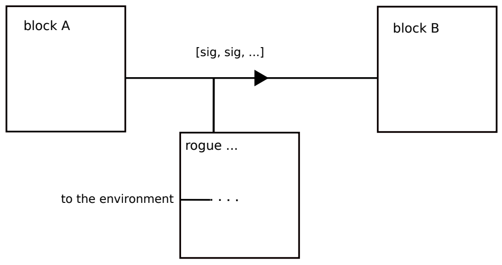
Figure 3a) shows an SDL 96 diagram illustrating channel substructure. Figure 3b uses a similar diagram to illustrate interference with the channel by an external source. The only difference between the rogue and the SDL 96 channel substructure is that the rogue has an external connection that ultimately leads to the system environment, whereas a channel substructure connected only to the two blocks connected to the channel. The rogue is transformed in the same way as a channel substructure was transformed in SDL 96, leading to the structure shown in Figure 2.

The benefit of this is that the rogue is clearly identifiable, and its removal is likely to be less error-prone than removal of the rogue block in Figure 2.

To complete a model of rogue behaviour, a way for the rogue to ‘spoof’ the value of sender would also be needed. That is, a signal sent by the rogue would differ from the old channel substructure in that it would masquerade as the block from which signals were placed on the original channel. This represents a significant departure from normal SDL behaviour, and would require allowing the



(a) An SDL 96 channel substructure



(b) A rogue interferes with a channel

Fig. 3: A rogue block can be represented like an old SDL 96 channel substructure

anonymous variable whose value is accessed by **self** to be updated by processes in a rogue block. However, the benefits of being able to model unwanted behaviour in this way, and then to remove the unwanted behaviour safely before deploying the new system indicate that this change is worth further consideration.

4 Multiple recipients

Different modelling formalisms have different approaches to handling events. In SDL 2010, an event, modelled as a signal, is handled by a single agent selected from a set of potential recipients [2]. This contrasts with Harel state machines in which a signal is handled by every agent that is capable of receiving it [15].

Ambient and broadcast communications characterise the Internet of Things. For example, the smart city broadcasts information about atmospheric conditions to all users who wish to receive that information in order to avoid areas with high levels of allergens. Also, some activities require different agents to coordinate their response to signals. For example, in the sheep monitoring investigation, the position of each sheep was determined by triangulation, so more than one recipient had to react to the sheep's beacon.

However, event sharing also leads to potential conflict between the different agents that could potentially react to an event [16]. For example, when sensors in the programmable city detect road traffic congestion, a coordinated response is needed to create appropriate diversions, and road safety will dictate that that response should be controlled by a single agent[3].

4.1 Using SDL 2010 to model multiple recipients

Sending signals to multiple known recipients Figure 4 illustrates a block that sends the same signals to three other blocks. The model illustrated in Figure 5 uses an intermediate block to achieve the same communication between block A and blocks B, C and D. The second approach has the advantage that all the logic relating to copying and forwarding signals is contained in the intermediate block.

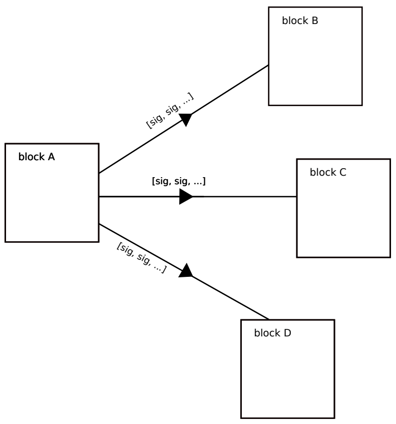


Fig. 4: Signals are sent directly to known recipients

Represent a broadcasting channel as a signal repository A variation on the approach illustrated in Figure 5 can be used to model broadcast wireless communications. Signals on a broadcast wireless channel can be accessed by any device that is tuned to that channel. This can be modelled in SDL by defining a block to represent the wireless channel. The broadcasting agent sends signals to that block via a non-delaying channel, and the block stores those signals, each with a time-stamp indicating when the signal arrived. Other agents send requests to the block which retrieves and delivers the stored signals. Constraints governing the usability of signals apply and old signals are eventually discarded as they expire.

[3] It is conceivable that roads and junctions, or vehicles, could negotiate diversions, but establishing that this was safe and effective would require further research.

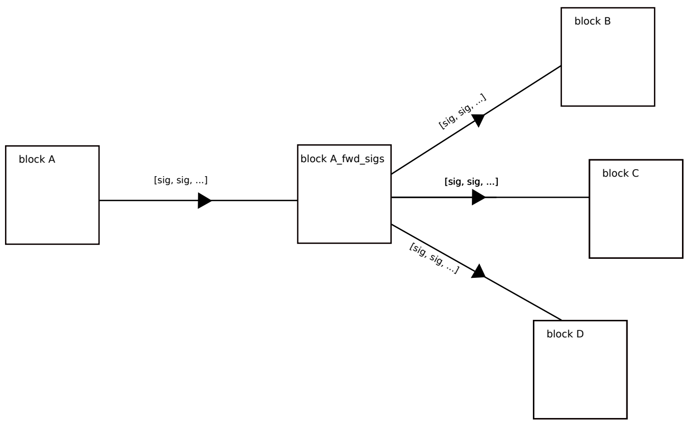


Fig. 5: Signals are sent to an intermediate block, which copies the signals to their recipients

4.2 Possible extensions to SDL

The approaches outlined above all conform to the SDL principle that each signal is consumed by a single agent. Reaction by a single agent leads to robust systems in which a single recipient modelled as a state aggregation coordinates the actions of all the agents, modelled as state partitions, that should respond to an event. However, it also leads to rather rigid systems, in which formation and dissolution of ad-hoc collections of cooperating or competing agents cannot be modelled directly.

Some approaches to providing flexible, controllable signal handling in SDL are discussed below.

Allow the sender to specify multiple recipients In SDL 2010, the sender of a signal can specify a recipient. This could be extended to allow the sender to specify that more than one recipient should respond to the signal. This would allow an IoT developer to specify the relationship between the sender and recipients directly, without the need to also specify a separate agent to replicate signals for each recipient.

This could be implemented in SDL 2010 by introducing transformations that inserted a new block between the sender of the signal and its intended recipients. Adding new channels from the new block to the recipients, and re-naming the signals would complete the transformation.

This change does not require any change to basic SDL, but could be implemented as a syntactic extension.

Allow agents to indicate that they will always respond to a signal A signal recipient could specify that it always responds to a signal. This could be implemented in a similar way to the previous suggestion, with each recipient receiving its own version of the signal. This aproach also has the advantage that it facilitates model re-use, as it allows an existing model to be encapsulated and extended by the addition of new signal recipients.

The intermediate block Both these approaches make use of an intermediate block, as in Figure 6, that duplicates signals sent by an originator to multiple intended recipients. This block either acts as a server, accepting requests from those recipients and treating stored signals as data to be forwarded to recipients on demand, or as a router, that forwards copies of the original signal to individual recipients.

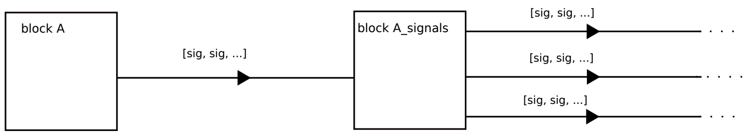


Fig. 6: An intermediate block makes signals available to different recipients

As illustrated in Figure 5, an intermediate block be modelled directly using SDL 2010 without modification. Alternatively, it would be useful to introduce some convenient syntax that would cause the block to be introduced by transformation. Further consideration is need to decide which, if any, syntactic modifications of SDL are desirable.

5 Next steps

The engineer who wishes to model and simulate new systems for deployment in the IoT faces some specific communications challenges. Some of these were explored in the previous sections, and adaptations of SDL 2010 with a view to making it easier to meet those challenges were discussed.

The challenges, originally identified in [1], represented areas where SDL appeared to be less than ideally suited to modelling situations that are likely to occur in the IoT. However, for the most part, these can be addressed with fairly minor changes to SDL 2010.

Simulating system behaviour on a busy communications network can be modelled by providing a runtime library of delay functions that delay signals on a channel by a duration that depends on the signals already in the channel's delay queue. A minor change to SDL 2010 would make these available to the engineer using SDL. Further consideration needs to be given to the functions to be provided, but probability distributions, or cumulative distributions are likely candidates.

Undesirable interactions with external agents can already be modelled by representing those agents as part of a new IoT system. Re-instatement of a construct similar to the SDL 96 channel substructure, would make it possible to do this in a way that clearly indicates the external agent and that facilitates its removal prior to deployment.

Allowing signals to be handled by multiple recipients can also be achieved by adding an intermediate block between the sender and the recipients. Syntactic constructs to be transformed so as to introduce such a block could be defined, but further discussion and exploration is needed to identify which constructs are likely to be of interest.

As a final word of caution, past experience has indicated that added constructs do not always live up to their original promise. For example, the original channel substructure construct was never used. This may have been because detailed control of channel behaviour was not actually needed, or because tool support was never provided for the construct. Before introducing changes to SDL 2010, careful evaluation of potential tool support and of demand for the proposed modifications is needed.

References

1. Sherratt, E., Ober, I., Gaudin, E., Fonseca i Casas, P., Kristoffersen, F.: SDL - The IoT Language. In: Fischer, J., Scheidgen, M., Schieferdecker, I., Reed, R. (eds.) SDL 2015: Model-Driven Engineering for Smart Cities. LNCS 9369, pp 27–41, Springer International Publishing Switzerland (2015)
2. ITU-T: Z.100 series for SDL 2010, International Telecommunications Union 2011-15
3. Williams, A., Probert, R., Li, Q., et Al.: The winning entry in the SAM 2002 design contest. In: Reed, R., Reed, J. (eds.) Proc. SDL 2003, LNCS 2708, pp 387–403, Springer-Verlag, Berlin, Germany (2003)
4. Ellsberger, J., Hogrefe, D., Sarma, A.: SDL: Formal Object-oriented Language for Communicating Systems. Prentice Hall Europe (1997)
5. Metzger, A.: Feature interactions in embedded control systems. Computer Networks 45, 625–64 Elsevier (2004)
6. Bozga, M., Graf, S., Mounier, L., Ober, I., Roux, J.-L., Vincent, D.: Timed Extensions for SDL. In: Reed, R., Reed, J. (eds.) Proc. SDL 2001: Meeting UML: LNCS 2078, pp 223–240, Springer Berlin Heidelberg (2001)
7. Bozga, M., Fernandez, J.-C., Ghirvu, L., Graf, S., Krimm, J.-P., Mounier, L., Sifakis, J.: IF: An Intermediate Representation for SDL and its Applications. In: Dssouli, R., Bochmann, G., Lahav, Y. (eds.) SDL'99 - The Next Millennium, Proc. of the Ninth SDL Forum, Montreal, June 1999, Elsevier (1999)
8. Graf, S.: Expression of Time and Duration Constraints in SDL. In: Sherratt, E. (ed.) Proc. SAM 2002: Telecommunications and beyond: The broader applicability of SDL and MSC: LNCS 2599, pp 38–52, Springer Berlin Heidelberg (2003)
9. Prinz, A.: SDL Time Extensions from a Semantic point of View. In: Sherratt, E. (ed.) Proc. SAM 2002: Telecommunications and beyond: The broader applicability of SDL and MSC: LNCS 2599, pp 38–52, Springer Berlin Heidelberg (2003)
10. Blanchard, T.: Endocrine Inspired Control of Wireless Sensor Networks: Deployment and Analysis Aberystwyth University, PhD Thesis (2016)
11. Kelly, B., Crowther, M., King, J.: Feature interaction detection using SDL models. In: IEEE GLOBECOM, 1994, Vol.3, pp.1857–1861

12. Turner, K.J.: Formalising Graphical Service Descriptions using SDL. In: Reed, R., Reed, J. (eds.) Proc. SDL 2003, LNCS 2708, pp 183–202, Springer-Verlag, Berlin, Germany (2003)
13. Chan, K.Y., Bochmann, G.V.: Methods for designing SIP services in SDL with fewer feature interactions. In: 7th Feature Interactions in Telecommunications and Software Systems, pp 59–76, IOS Press (2003)
14. Doldi, L. SDL Illustrated: Laurent Doldi (2001)
15. David Harel, Yishai Feldman: Algorithmics: the Spirit of Computing, Pearson Education Limited, 2004
16. Sherratt E.: SDL in a Changing World. In Amyot, D., Williams, A.W. (eds.) System Analysis and Modeling: 4th International SDL and MSC Workshop, SAM 2004, LNCS 3319, Springer (2005)