Asynchronous modeling in railway systems

Emmanuel Gaudin
PragmaDev, France
emmanuel.gaudin@pragmadev.com

Abstract: *Models in the railway industry are often based on synchronous technologies such as Matlab or Scade. This is due to technical reasons, but because of its concepts the abstraction level of synchronous models are very low and very close to the implementation level. A serious gap is observed between the requirements described in natural textual language and the model which is basically an implementation. The increasing level of system complexity, combining communicating subsystems, calls for a more abstract model. This paper will first discuss why synchronous technologies have been used in this type of systems, then an experiment of using an asynchronous technologies on a real ERTMS case coming from SNCF is described, and finally the paper will conclude on how an asynchronous modeling technologies could make the link between the informal textual requirements and the implementation of the system.*

Keywords*: Modeling, Asynchronous, Synchronous, Matlab, Lustre, SDL, TTCN-3, Railways, ERTMS*

Introduction

When it comes to modeling two main questions have to be addressed. The first one is about positioning the model in the development cycle ; defining if the model is a requirement, a specification, or a design. The second one is about the modeling technology to use ; depending on what the model is aiming at. The lower is the model level, the more specialized is the modeling technology, and the narrower is the scope of the model.

In [1] the technologies usually applied to model train systems are listed such as the B method, Scade, Simulink/Stateflow. In [2] and [3] the authors present how they have written a specific type of model in order to verify specific safety properties. The models are usually dedicated to the targeted model checking technology and can not be used for anything else.

In [4] is presented the work done by SNCF to verify safety rules using The Mathworks tools.

In [5] the author presents a tool that makes a link between a system level model written with Papyrus SysML modeler and a design level model written with SCADE Suite.

In [6], following the ASSERT FP6 european project, the European Space Agency has been promoting the TASTE (The ASSERT Set of Tools for Engineering) framework. Because each technology is best suited for a part of the overall system, TASTE framework aims at gathering the different technologies in a consistent framework. The top level model is an architecture model based on AADL and ASN.1. The different AADL architecture blocks are further developed with a dedicated technology such as Scade or SDL. When all models are validated a code generator automatically gathers the code generated by the different tools.

In the above references, the choice of the modeling language is often driven by the possible verification associated to the technology. For that purpose models are based on low level modeling technologies that are very close to the implementation details.
Attempts to raise up the modeling level have been done using a combination of languages. For that purpose the synchronous or asynchronous approaches are put at the same level and a third language is used as an overall model view (SysML or AADL). In this paper we are experimenting a different approach in which an asynchronous SDL model is used as a bridge between the requirements and a low level synchronous model. To demonstrate this, an existing Matlab model is taken as an example and translated to an SDL model. Using an SDL simulator and solver the system functions are then analyzed. Finally there will be a discussion on what the SDL model brings to a Matlab model.

A natural synchronous approach

In the old days, train systems were exchanging simple information such as “is the train present on that portion of track” , “are the doors open or not”. These information can be detected with simple electrical detectors along the tracks, on the platforms, or in the train itself. These detectors behave like electrical switches and the information can be extracted from the fact that the circuit is open or closed. From a software point of view, all this information can be represented by binary variables. The rationale is a logical combination of the different information gathered. It would be something like “if the train is not stopped at the platform, then the doors should be closed”. And this information would be verified every time some new information was received from the sensors. Each clock tick, the information from the sensors is gathered and given to a logical system that will compute all the entries and produce some outputs.
That is the basic principle of a synchronous approach. All the entries are valid at the same time, some logical operators combine the inputs taking into consideration the previous values of the inputs, and produces some outputs. Each clock tick, the whole information is computed again and again. The fact that the system re-computes everything at every clock tick actually reduces the possible number of cases that might occur in the system. It is therefore much easier to verify properties at each clock tick and make sure the system behaves properly whatever happens, whatever the information read by the sensors.

New approach for upcoming systems

Train management systems are evolving, and in particular the European Rail Traffic Management System (ERTMS [7]), which was initiated in the 90’s by the European Union, aims at harmonizing and expanding the capabilities of train control systems over Europe so that a train crossing borders does not need a specific controller for each country it crosses. This standard covers specification of on board equipments, on the trackside equipments, as well as communication information systems. The information exchanged includes speed, acceleration and so on. It is more and more complex and is getting quite far from the original binary information. Furthermore all the equipments are not mechanically or electrically connected, they are now completely desynchronized from one another. The information is not that simple any more, it is complex and unpredictable. Using synchronous technologies might work on a local level but will definitely not be sufficient to describe new features that combine a lot of communication. In fact the higher is the level of the view, the less a synchronous description will fit. This is a known issue in systems where complexity is increasing. It has been theorized and discussed in several publication as the GALS (Globally Asynchronous Locally Synchronous [10]) theory of system description since the 80s. At that time, asynchronous descriptions were transformed to synchronous descriptions based on the theory that asynchronous models could be deconstructed to synchronous models and reconstructed back [11]. The point was to be able to use the existing and mature synchronous validation and verification technologies on the market at that time [12].
This works as long as the model is close to the implementation. This is not satisfying any more in complex communicating systems as the first thing to do when developing a new feature in a system is to verify its

functionality before trying to implement it. That raises the need for asynchronous descriptions and verification techniques.

An asynchronous description of existing models

On one side the higher the abstraction level is, the more asynchronous are the relations among the elements in the system. On the other side in order to produce a relevant and verifiable dynamic description, the model needs to be executable. That means it should be statically and dynamically unambiguous from a semantic point of view.
SDL [6] international standard that was initially designed to describe telecommunication protocols is a good candidate for this type of description. It is by nature asynchronous, it combines graphical views for architecture and state machines, and includes an action language with simple data types to describe a detailed behavior whenever needed in the description.

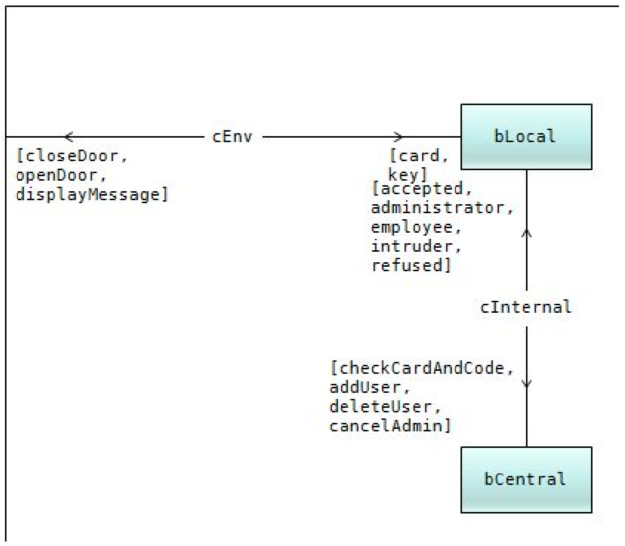

cEnv
[closeDoor, openDoor, displayMessage]
[card, key]
bLocal
[accepted, administrator, employee, intruder, refused]
cInternal
[checkCardAndCode, addUser, deleteUser, cancelAdmin]
bCentral

Figure 1: A basic SDL architecture

Figure 1 shows an SDL architecture with two blocks. Each block can be further decomposed in sub-blocks. At the lower level of the architecture one or several finite state machines describe the behavior of its container. The flow of information between the blocks is message based. Each state machine has its own implicit FIFO message queue.There is no clock based inputs in an SDL system. Only the sequence of events matters.

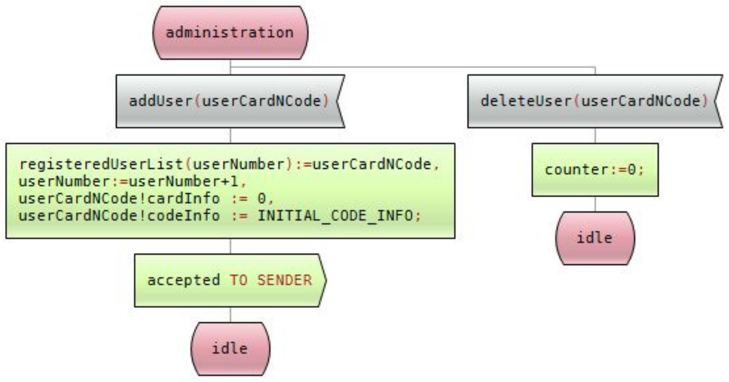

administration
addUser(userCardNCode)
deleteUser(userCardNCode)
registeredUserList(userNumber):=userCardNCode,
userNumber:=userNumber+1,
userCardNCode!cardInfo := 0,
userCardNCode!codeInfo := INITIAL_CODE_INFO;
counter:=0;
accepted TO SENDER
idle
idle

Figure 2: A simple SDL state machine

Figure 2 shows a simple SDL state machine. In the *administration* state, the state machine will read its message queue. If *addUser* message is received the instructions below the *addUser* input symbol are executed, an *accepted* message is sent to the sender of the *addUser* message, and the transition ends back in the *idle* state. If message *deleteUser* is received, the counter internal variable is set to 0, and the transition ends also in the *idle* state.
The different state machines in an SDL model run in parallel. The main issue with this type of description is verification. Because of its asynchronous nature, events can occur at any time, independently from each other, and this creates a huge number of possible scenarios. Model checking tools can explore the possible combinations, but the number of cases is sometimes very difficult to handle making verification of properties on this type of system a real challenge.
Since this type of description is well suited for a high level description it is naturally close to a functional description or a high level requirement. It is therefore quite interesting to analyze how the requirements could be translated into an asynchronous model and a synchronous model, and see if one could be translated to the other. This is what was done on a Radio Block Center from the ERTMS[7]. Figure 3 and 2 show the architecture level using Matlab and using SDL. Even though both model contain the same blocks the main difference is the communication semantic. Information exchange is synchronous in Matlab and message based in SDL.

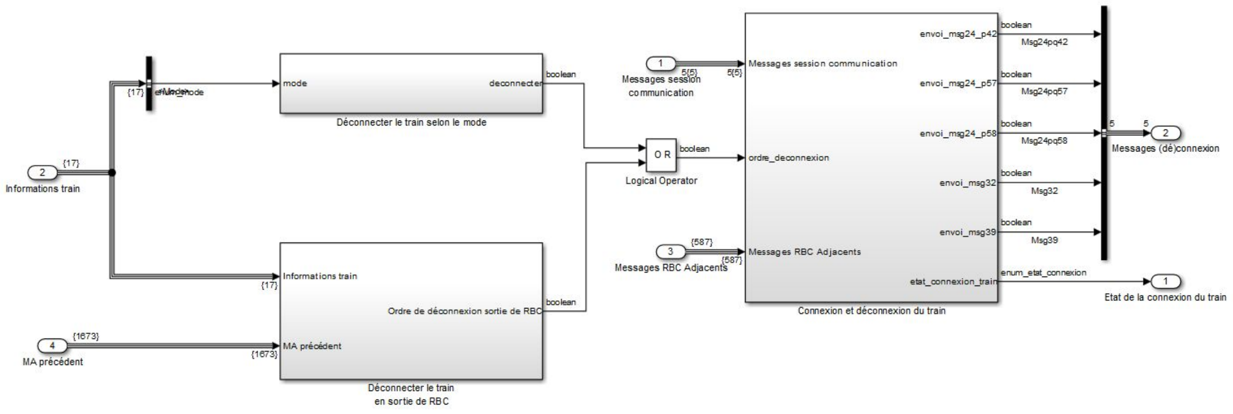


Figure 3: Architecture described with Matlab

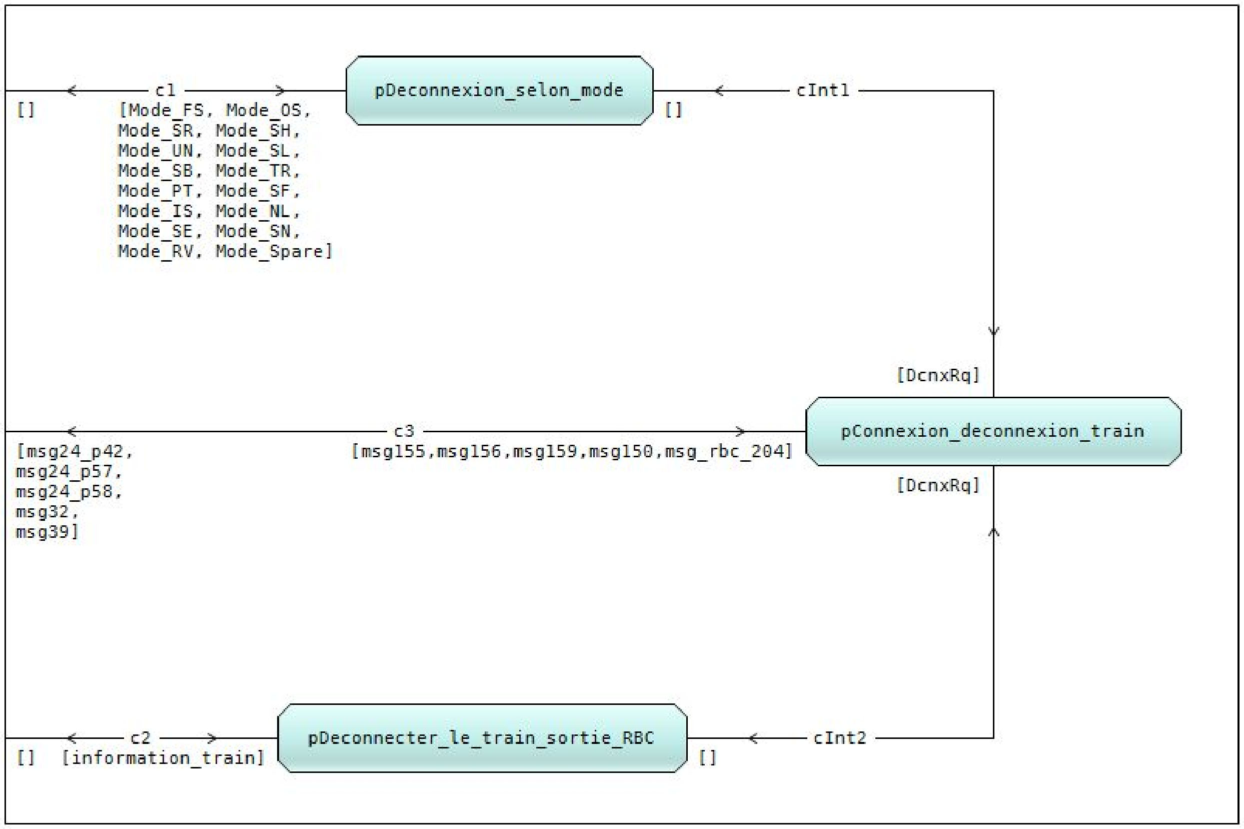

pDeconnexion_selon_mode
c1
[]
[Mode_FS, Mode_OS, Mode_SR, Mode_SH, Mode_UN, Mode_SL, Mode_SB, Mode_TR, Mode_PT, Mode_SF, Mode_IS, Mode_NL, Mode_SE, Mode_SN, Mode_RV, Mode_Spare]
cInt1
[]
[DcnxRq]
pConnexion_deconnexion_train
[DcnxRq]
c3
[msg24_p42, msg24_p57, msg24_p58, msg32, msg39]
[msg155,msg156,msg159,msg150,msg_rbc_204]
pDeconnecter_le_train_sortie_RBC
c2
[]
[information_train]
cInt2
[]

Figure 4 - Architecture in SDL with three state machines

Matlab diagram in Figure 5 indicates the description of the block is done with a state machine. It lists all the inputs and outputs of the state machine. This is not necessary in SDL as a process behavior is always described with a state machine.

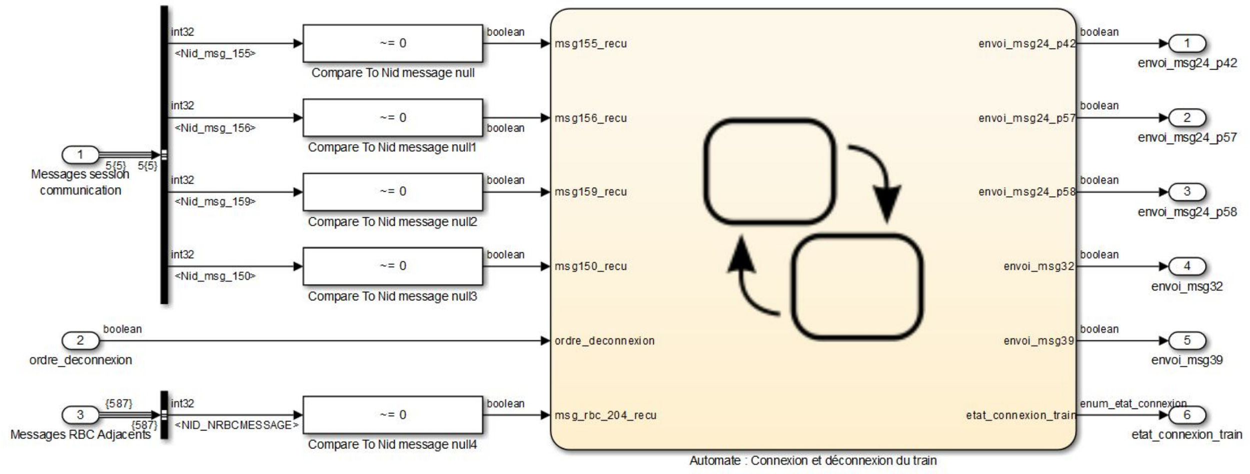
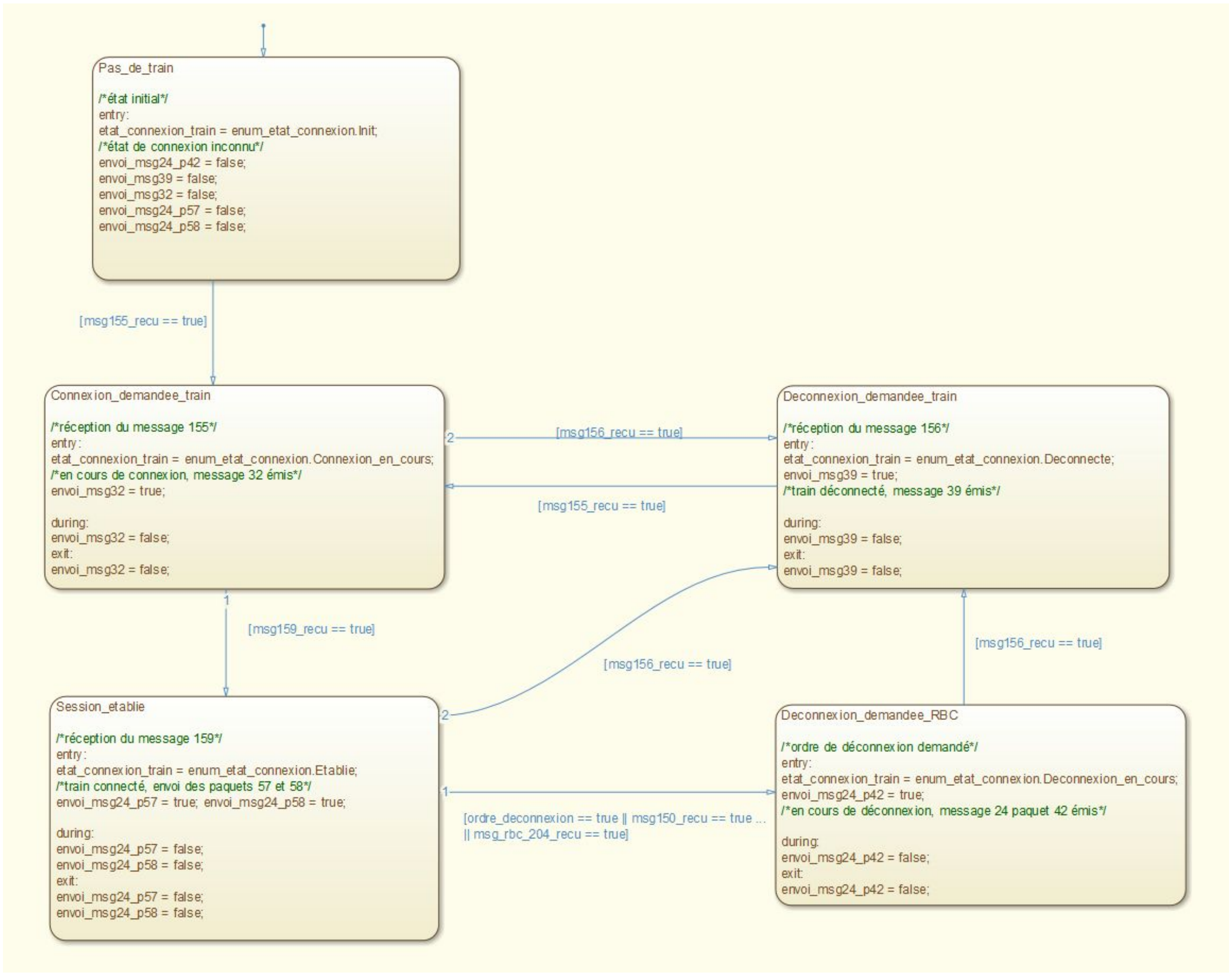


Figure 5 : Connect and disconnect block is made of one synchronous state machine

The Matlab state machine is described in Figure 6 and the SDL equivalent state machine is described in Figure 7.



```
Pas_de_train

/*état initial*/
entry:
etat_connexion_train = enum_etat_connexion.Init;
/*état de connexion inconnu*/
envoi_msg24_p42 = false;
envoi_msg39 = false;
envoi_msg32 = false;
envoi_msg24_p57 = false;
envoi_msg24_p58 = false;
```

```
Connexion_demandee_train

/*réception du message 155*/
entry:
etat_connexion_train = enum_etat_connexion.Connexion_en_cours;
/*en cours de connexion, message 32 émis*/
envoi_msg32 = true;

during:
envoi_msg32 = false;
exit:
envoi_msg32 = false;
```

```
Deconnexion_demandee_train

/*réception du message 156*/
entry:
etat_connexion_train = enum_etat_connexion.Deconnecte;
envoi_msg39 = true;
/*train déconnecté, message 39 émis*/

during:
envoi_msg39 = false;
exit:
envoi_msg39 = false;
```

```
Session_etablie

/*réception du message 159*/
entry:
etat_connexion_train = enum_etat_connexion.Etablie;
/*train connecté, envoi des paquets 57 et 58*/
envoi_msg24_p57 = true; envoi_msg24_p58 = true;

during:
envoi_msg24_p57 = false;
envoi_msg24_p58 = false;
exit:
envoi_msg24_p57 = false;
envoi_msg24_p58 = false;
```

```
Deconnexion_demandee_RBC

/*ordre de déconnexion demandé*/
entry:
etat_connexion_train = enum_etat_connexion.Deconnexion_en_cours;
envoi_msg24_p42 = true;
/*en cours de déconnexion, message 24 paquet 42 émis*/

during:
envoi_msg24_p42 = false;
exit:
envoi_msg24_p42 = false;
```

Figure 6 : Matlab synchronous state machine

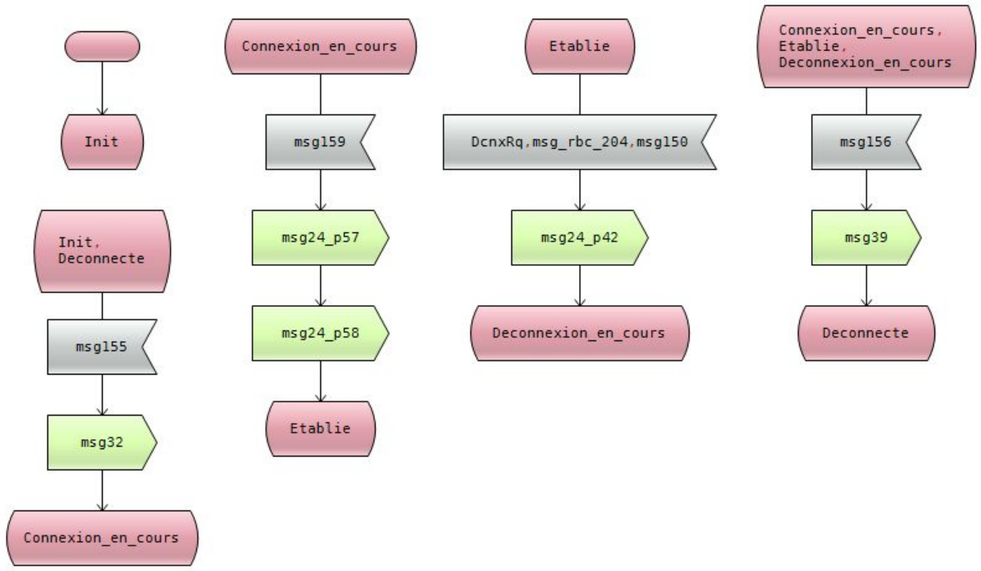


Figure 7 : SDL asynchronous state machine

In the example described in Figure 6 & 7 the inputs are very similar. For example reception of msg155 event is done setting the boolean variable msg155_recu to true in Matlab, while it uses the message input symbol in SDL. The Matlab representation forces the modeler to make sure msg155_recu is set to false after being received because if not it might be taken into consideration again in another transition. Similarly to output some information from the state machine, the Matlab model sets boolean values to true or false. For example in the Session_etablie state, envoi_msg32 is set to true when entering the state, then set to false while in the state, and again set to false when exiting the state. In an event based language such as SDL, there is only one msg32 output when msg155 is received and that's it. In that sense it makes things much clearer.

The other example below, Deconnecter_selon_mode in Figure 8 & 9, shows how to disconnect the train depending on the mode in which it is.

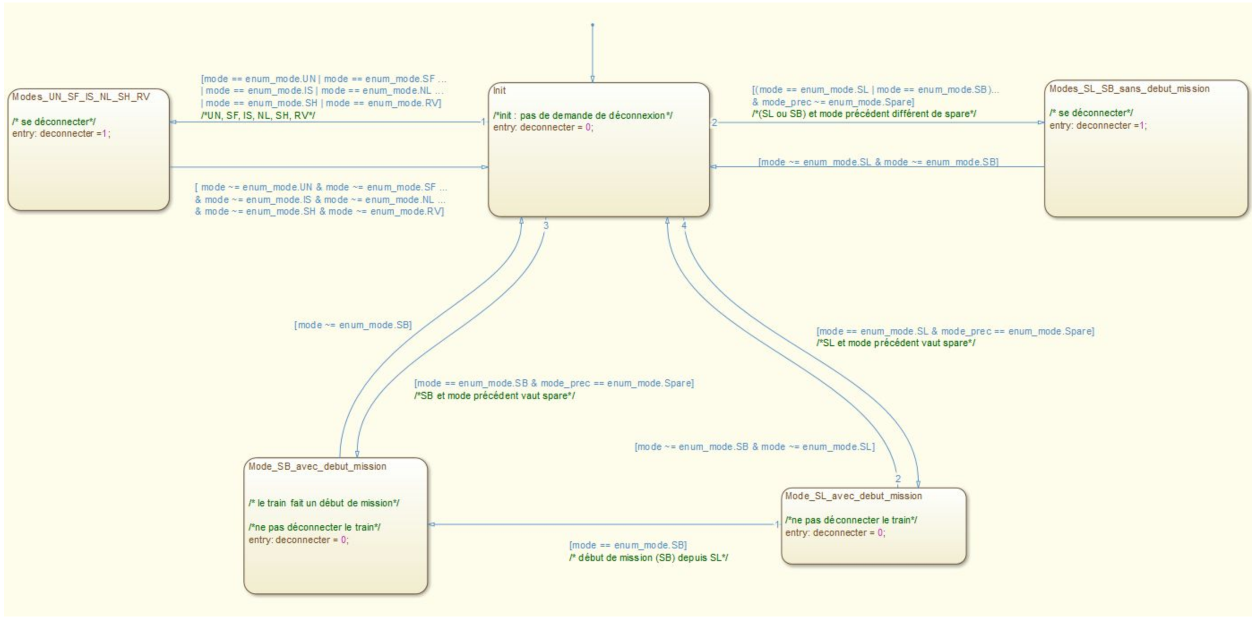

Modes_UN_SF_IS_NL_SH_RV
/* se déconnecter*/
entry: deconnecter =1;
[mode == enum_mode.UN | mode == enum_mode.SF ...
| mode == enum_mode.IS | mode == enum_mode.NL ...
| mode == enum_mode.SH | mode == enum_mode.RV]
/*UN, SF, IS, NL, SH, RV*/
[mode ~= enum_mode.UN & mode ~= enum_mode.SF ...
& mode ~= enum_mode.IS & mode ~= enum_mode.NL ...
& mode ~= enum_mode.SH & mode ~= enum_mode.RV]
Init
/*init : pas de demande de déconnexion*/
entry: deconnecter = 0;
1
2
3
4
[(mode == enum_mode.SL | mode == enum_mode.SB)...
& mode_prec ~= enum_mode.Spare]
/*(SL ou SB) et mode précédent différent de spare*/
[mode ~= enum_mode.SL & mode ~= enum_mode.SB]
Modes_SL_SB_sans_debut_mission
/* se déconnecter*/
entry: deconnecter =1;
[mode ~= enum_mode.SB]
[mode == enum_mode.SB & mode_prec == enum_mode.Spare]
/*SB et mode précédent vaut spare*/
[mode == enum_mode.SL & mode_prec == enum_mode.Spare]
/*SL et mode précédent vaut spare*/
[mode ~= enum_mode.SB & mode ~= enum_mode.SL]
2
Mode_SB_avec_debut_mission
/* le train fait un début de mission*/
/*ne pas déconnecter le train*/
entry: deconnecter = 0;
Mode_SL_avec_debut_mission
/*ne pas déconnecter le train*/
entry: deconnecter = 0;
1
[mode == enum_mode.SB]
/* début de mission (SB) depuis SL*/

Figure 8 - "Disconnect depending on the mode" Matlab state machine

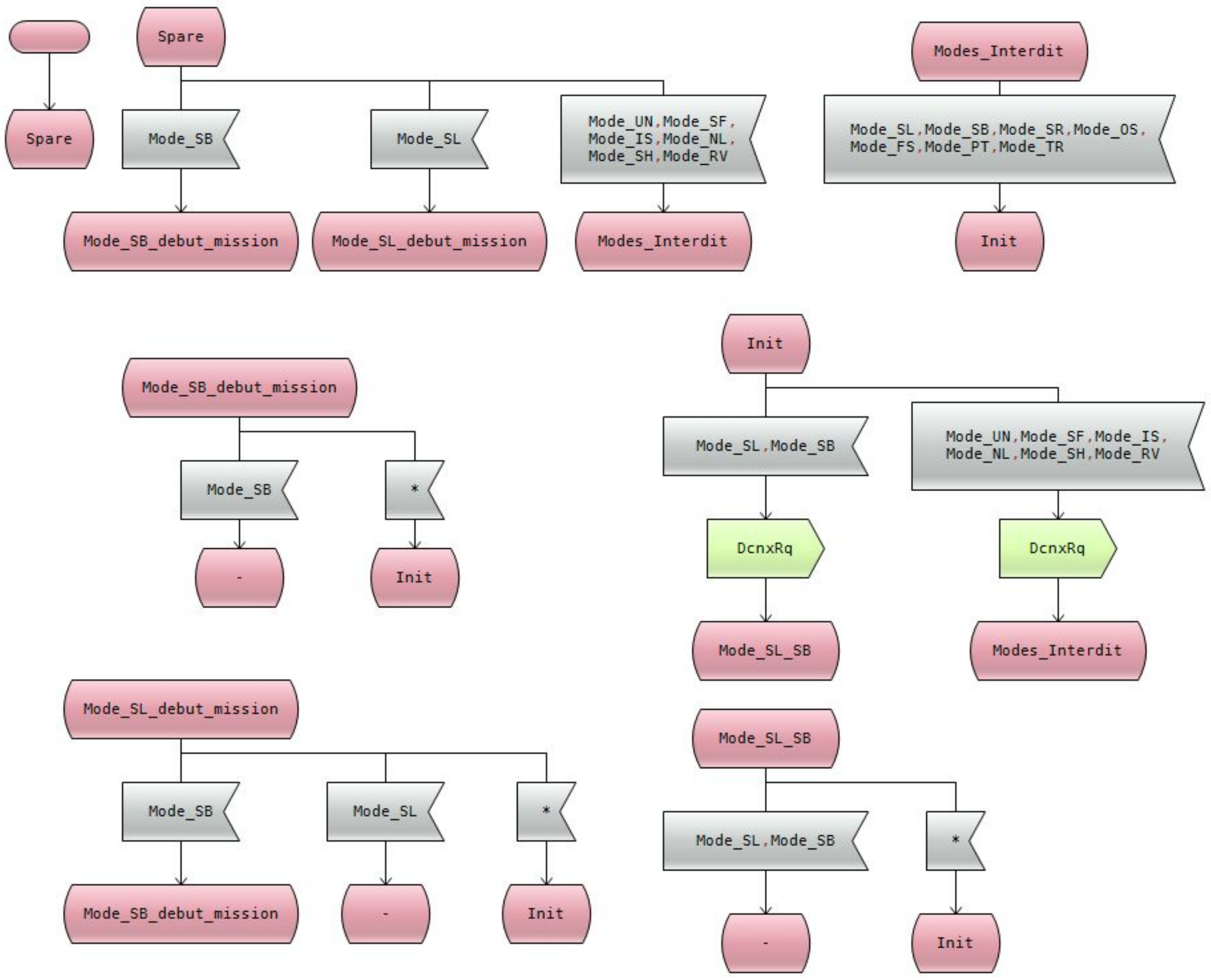


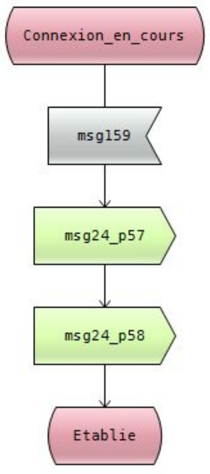
Figure 9 - "Disconnect depending on the mode" SDL state machine

In that example the main difference is on the inputs of the state machine. The Matlab model uses logical operators AND and OR to identify which input was received; the SDL model is just a list of inputs, and the star means any other input.

In both examples the model is equivalent from a functional point of view, depending on the reader's technical background one or the other might be easier to read and understand.

Model verification

The experiment included some simulation of the SDL model with small prototyping graphical user interface in order to verify the behavior was correct. Once the model was considered correct, PragmaDev symbolic resolution tool, result of PragmaList [8][9] common lab, has been experimented on the model. This technology combines the transitions from a symbolic point of view, and tries to solve each possible combination like it would do with an equation. If there is a solution to the equation the path is possible. The first objective with that technology was to automatically generate the minimum number of test cases with a maximum coverage. After a few trials the tool could not reach two transitions in the model. A manual analysis rapidly concluded this use case did not allow one of the generic functions in the model to return the values required to reach these two transitions. Once this was settled test generation out of the model was



successfully experimented and 17 test cases covering all transitions were automatically generated.
Five properties have been written to be verified on the model. As for the experiment, the properties were actual pieces of the state machine written with another language. For example the first property verifies that when in state Connexion_en_cours, when receiving msg159 the state machine goes to state Etablie and not any other.
The symbolic resolution tool has been ran on the model with its properties for a few hours reaching a substantial depth of search, meaning a substantial number of transition combination. As a result, within this exploration perimeter the properties were satisfied.

How to link asynchronous models to synchronous models

During the experiment it has been established the SDL model was further away from the implementation than the Matlab one. Because of its asynchronous principles it was more of a functional view of the behavior and therefore closer to the requirements. This clearly validated the idea of having a high level asynchronous executable functional model to make sure the requirements are properly understood. The question was how to link this asynchronous approach to a synchronous one. It turned out an asynchronous model, including a test case, can easily be connected to a synchronous one. For example a synchronous input can be evaluated at each tick and when the value of the input changes it generates an asynchronous message (Figure 10). On the other way around an asynchronous message output can be converted to a clock based value.

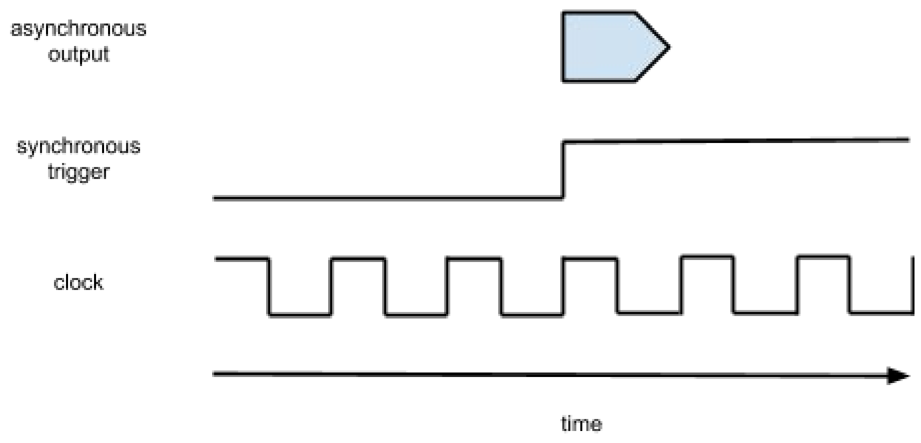


Figure 10: An synchronous change of value can be transformed to an asynchronous send

This shows that it would be possible to generate code out of the asynchronous model and connect to a synchronous target, or to generate test cases out of the asynchronous model and run them against a synchronous implementation in order to check it is conform to the model.

Conclusion & Future work

The experiment on this real use case in the railway domain has demonstrated that an SDL executable asynchronous model could be functionally equivalent to a Matlab synchronous model. Because of its asynchronous nature the SDL model is closer to the requirements, where a Matlab model is closer to the implementation. An SDL model could therefore be used by stakeholders early in the development process to formalize requirements and to verify them from a functional point of view. A Matlab model would still be used later on for the implementation. And the SDL model would be the reference to verify functional properties, or to generate test cases to verify the final implementation is functionally conform to the initial requirements.

Bibliography

[1] Flammini, F., "Railway Safety, Reliability, and Security: Technologies and Systems Engineering", IGI Global, May 31, 2012 - Technology & Engineering - 487 pages.
[2] Liu, Jiang (et al.), "A Calculus for Hybrid CSP", Programming Languages and Systems 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010 Proceedings, Springer LNCS 6461.
[3] Cimatti, Alessandro (et al.), "Formal Verification and Validation of ERTMS Industrial Railway Train Spacing System", Computer Aided Verification, 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings, Springer LNCS 7358.
[4] Callet, S., el Fassi, S., Fedeler, H., Ledoux, D. and Navarro, T. (2014) The Use of a "Model-Based Design" Approach on an ERTMS Level 2 Ground System, in Formal Methods Applied to Industrial Complex Systems (ed J.-L. Boulanger), John Wiley & Sons, Inc., Hoboken, NJ, USA.
[5] Le Sergent T., "SCADE A comprehensive framework for critical system and software engineering", SDL Forum 2011, Springer LNCS 7083.
[6] International Telecommunication Union: Recommendation Z.100 (12/11) Specification and Description Language (SDL). http://www.itu.int/rec/T-REC-Z.100
[7] European Rail Traffic Management System ERTMS, http://www.era.europa.eu/Core-Activities/ERTMS/Pages/home.aspx
[8] Gaudin E., Deltour J., Faivre A., Lapitre A., "Model-Based Testing: An Approach with SDL/RTDS and DIVERSITY". System Analysis and Modeling: Models and Reusability. 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings. Editors: Amyot, Daniel, Fonseca i Casas, Pau, Mussbacher, Gunter (Eds.). Springer LNCS 8769.
[9] www.pragmalist.org
[10] "Globally asynchronous locally synchronous",Wikipedia, http://en.wikipedia.org/wiki/Globally_asynchronous_locally_synchronous
[11] A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J. C. M. Baeten and S. Mauw, editors, Proceedings of CONCUR'99, volume 1664 of LNCS, pages 162-177. Springer, 1999.
[12] M. Mousavi , P. L. Guernic , J.-P. Talpin , S. Shukla and T. Basten "Modeling and validating globally asynchronous design in synchronous frameworks", Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, pp.384 -389.