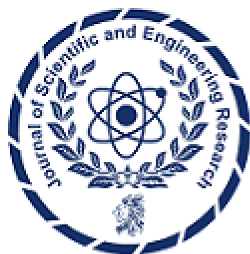
*Available online www.jsaer.com*

***Journal of Scientific and Engineering Research*, 2020, 7(2):247-256**




**Research Article**

**ISSN: 2394-2630**
**CODEN(USA): JSERBR**

# Translating GRAFCET Charts into C++ Code for Embedded Control Applications

**Médésu Sogbohossou*, Clément Hounkpevi**

Laboratoired'Électrotechnique, de Télécommunications et d'Informatique Appliquée (LETIA), École Polytechniqued' Abomey-Calavi (EPAC), UAC, 01 BP 2009 Cotonou, Benin
*Corresponding author: E-mail: medesu.sogbohossou@epac.uac.bj

**Abstract** For the implementation of an embedded application, the specification phase of the system allows it to be validated. A GRAFCET chart (IEC 60848) may be the modelling language for this kind of systems. This paper introduces the translation of GRAFCET chart into a code for embedded platforms in order to perform control applications. Here, GRAFCET charts are edited with Jgrafchart software, which returns a XML file that is used to translate useful information of the chart into code in C++ language, particularly for mbed microcontrollers. The notable elements of GRAFCET that we consider are steps (and their corresponding actions), transitions, variables and especially elements that take into account the hierarchical aspect in GRAFCET such as macro-steps and enclosing steps.

**Keywords** GRAFCET (IEC 60848), code generation, control applications, mbed platform, Jgrafchart

## 1. Introduction

A good design of a complex, reliable and scalable computer system requires the use of a development cycle model. Thus, a specification phase must precede the detailed design and development phases. The specification phase allows for the description of the system to be conceived by formal abstractions enabling the verification of the expected properties (via simulations, mathematical proofs, model-checking...) [1]: this offers some guarantee of proper functioning before the implementation of a critical application. Many researchers try to meet this requirement. For instance, M. Samek [2] uses UML State- charts as the specification language and develops tools for automatic code generation for embedded platforms. More recently, the work in [3] presents a tool (named PN2A) for the translation of time Petri net models into compilable code under Arduino; however, hierarchical modelling which is essential for complex systems is not taken into account.

In the domain of control applications, the GRAFCET standard [4] (Acronym in French: *GRAphe Fonctionnel de Commande Etape Transition*) is a semi-formal specification language for the functional description of the behaviour of the sequential part of a system. It is comparable to SFC standard [5] (*Sequential Function Charts*) intended to implement a system for Programmable Logic Controller (PLC). The main difference between the two standards is about how computing an evolution. GRAFCET standard is event-driven: every input change starts a sequence of transition firings until a stable situation is reached, before another input change. Whereas SFC standard is clock-driven: all inputs of the PLC are scanned at every tick and then may cause only one set of simultaneous firings; but several interpretations are conceivable about when updating variables. Many works (such as [6]) are interested in generating code from the SFC implementation language for PLC. The advantages of the GRAFCET standard are to be more formally defined than SFC, and to allow different levels of modelling (of growing complexity) before the implementation: a system may be formally validated by model-checking on

these gradual models [7, 8]. This standard is also less specific to PLC issue. It can also be used to implement control applications as well as the SFC standard.

In our context, the only known similar and more recent work is about C-code implementation from GRAFCET and concerns the reference [9]: a compiler (called GeCé) implements the methodology presented by the authors. Apart from the fact that the detailed algorithms used in the process of translating a grafcet into C-code are not presented, some shortcomings need to be noted, such as: the hierarchical concept of enclosures and macro-steps is not taken into account in [9], and evaluating transition conditions (called receptivities) only at the beginning of an iterated evolution is not strictly in accordance with the standard in case of edge variables. In the sequel, we propose more detailed algorithms and method of translation, and we take into account enclosures and macro-steps.

Among the modern platforms for the development of embedded systems, *mbed* is one of the most widespread [10]: *mbed* includes a real-time operating system (OS) called *mbed OS*. The software *JGrafchart* [11, 12] is a graphical GRAFCET chart editor (even if not completely dedicated to the GRAFCET standard) for implementation of sequential and control applications which produces a XML file. It is used here since an editor completely in conformity with the GRAFCET standard seems to not exist. The present work proposes a method to conceive a control application for an mbed platform from a GRAFCET chart (denoted *grafcet* in short) that describes the operations of this application. C++ code is produced for mbed from a grafcet edited under JGrafchart.

The next section of this paper presents a summary about GRAFCET and JGrafchart. In Section 3 is proposed the translation of elements of GRAFCET language into C++ code. Section 4 shows an application. Finally, Section 5 presents a summary of the results and formulates some perspectives.

## 2. Reminders

### 2.1. GRAFCET elements

A grafcet [4] is composed with three kinds of graphic elements: steps, transitions, and directed links.

A *step* models the state of the system. The set of active steps at a given moment constitutes the situation and the associated actions are executed. There are several types of steps, namely *macro-steps*, *enclosing steps* and *ordinary steps*. Moreover, a step may be initial or not except the case of a macro-step. The *actions* associated with a step are often actuator controls (motors, jacks). They can also be commands of auxiliary functions of automaton (counter, delay ...). We distinguish several types of actions such as continuous actions, conditional actions and stored actions:

- *continuous action*: It is executed if the related step is really active. A *conditional action* gets a logical proposition, called assignment condition, which can be true or false and can condition a continuous action;
- *stored action*: It makes it possible to assign a determined value to a variable. Different kinds of stored actions are distinguished: action on activation, action on deactivation, action on the clearing, and action on event.

A *transition* controls the move from one step to another, through the *transition condition* associated with this transition. The *transition condition* associated with a transition is a logical condition or an event. A logical condition is a boolean function of external variables and internal variables. An internal variable is related to the activations of the steps. An external variable is information coming from a sensor, a push button or an external system.

### 2.2. GRAFCET interpretation

Algorithm 1 [7] shows how a GRAFCET specification evolves from one situation to the next. At the line 1, actions on activation are done. After this, the loop (lines 2-15) is executed: at each iteration of the loop, all the transition conditions are evaluated at line 3 and the fireable transitions are computed. Then, at line 5, it is checked if exists at least one fireable transition and if applicable, these transitions are fired (line 6) and stored actions of the following steps are done (line 7). But if there is no fireable transition, the conditions of continuous actions are evaluated and these actions are executed if related condition is true.

```
1  Do the Stored Action on activation of the initial steps;
2  while True do
3      Evaluate condition of transitions;
4      Compute the set of fireable transitions;
5      if Exists a nonempty set of fireable transition then
6          Fire the transitions of the set (as one stage);
7          Do the Stored Actions;
8      else
9          Evaluate conditions of a Continuous Actions;
10         Set the Output allowed by the valid Continuous Action (and Reset the others);
11         Start or Restart the required clocks for the timed variables;
12         Wait for the occurence of some external event;
13         Get and latch all timer clocks;
14     end
15 end
```

**Algorithm 1.** *Execution scheme of a grafcet*

### 2.3. Editing GRAFCET charts with JGrafchart

It should be noted that no public tool is currently available to edit the grafcets; for instance in the work [9], no tool (even used privately) is shown up.

JGrafchart software is based on the specific GRAFCHART language: GRAFCHART is derived from several languages, including GRAFCET, SFC, Petri Net. In terms of similarity with GRAFCET, they have some common elements: the steps and the related actions, the transitions and the related conditions, the directed links, the AND divergences, the AND convergences, the variables. But, there are many GRAFCET elements that are not taken into account by JGrafchart: for instance, the semantics of enclosing steps is different from JGrafchart elements such as *Procedure steps* or *Process steps*. Nethertheless, we limit ourselves to a syntactic point of view to represent an enclosing step with JGrafchart.

A *Procedure* in JGrafchart (symbolized by a lonely rectangle) represents a partial grafcet. Here, an enclosing step is represented by a *Process step* with a corresponding Procedure. Each of the enclosures of the enclosing step is represented by an isolated macro-step (different from the common use of an ordinary macro-step) contained in the Procedure related to this enclosing step. And the sub-workspace of each such isolated macro-step contains the grafcet describing the intended enclosure.

## 3. Translation of GRAFCET elements for the mbed platform

The main elements of the GRAFCET language that we translate are variables, steps, transitions, transition conditions, actions, partial GRAFCET specifications (or partial grafcets) and, finally, the main process. There are three types of partial GRAFCET specifications: an ordinary partial grafcet, an expansion and an enclosure.

### 3.1. Variables

We define two types of variables including input/output variables and simple variables. The input/output variables are distinguished from the simple variables under JGrafchart by the fact that they are linked to the ports of the related microcontroller.

Figures 1, 2 and 3 present the representation in Jgrafchart and the translation into C++ code respectively of a digital input/output variable, an anolog input/output variable and a simple variable.

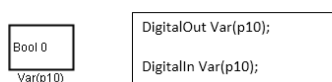



*Figure 1: Digital input/output Variable*

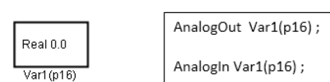



*Figure 2: Analog input/output Variable*

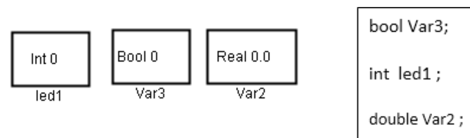



*Figure 3: Simple Variable*

### 3.2. Step

A step is considered as an object having some attributes namely : its types (ordinary or not, initial step, star step, input step or output step), its previous and current states, the real-time duration of activation of the step in order to possibly manage the timers linked to it, and an identifier. In addition to these attributes, a macro-step has a special state to allow the transitions that follow it to see it activated after the activation of the output step contained in its expansion. The Getter and Setter methods are used to manipulate these different attributes.

### 3.3. Transition

Just like a step, a transition is an object that has the following attributes: an identifier, the list of steps that follow this transition, the type of transition condition (either it is timed or not), the list of steps preceding this transition, a boolean variable of its validity and his fireability variable. Here again, Getter and Setter methods as well as other useful methods are used to manipulate the attributes mentioned above.

### 3.4. The condition of a transition

A transition condition is represented by a boolean function that returns the value "True" if the transition condition is true and "False" if the transition condition is false. The function is called each time one needs to test the transition condition. Moreover, it should be noted that transition conditions with delay are taken into account directly by the related transitions. Indeed, the time and the delay step are recorded as an attribute of the related transition. We present at Fig. 4 the translation of a transition condition into C++ code.

```
bool Receptivity_IdTransition(){
If(Wording) return 1;
return 0;
}
```

*Figure 4: Representation of a transition condition*

Here, the word "Wording" is used to name the transition condition.

### 3.5. Action

An action (Fig. 5) is a function that is executed by a process. When, it is a Continuous Action, it is the main process **main** (cf. Annex A.4) which executes the function when the grafcet is in the stable situation (no fireable transition). But in the case of Stored Action, it is the process that supports the partial grafcet (cf. Annexes A.1, A.2 and A.3) containing the step related to the action which takes this action in charge.

*Figure 5: Representation of Actions*

### 3.6. Partial grafcet

A partial grafcet is represented by a process (use of threads with mbed) that will execute a function related to this partial grafcet. The structure of this function depends on the type of partial grafcet (expansion for a macro-step, enclosure for an enclosing step, and the ordinary partial grafcet). A main process of the program is responsible for managing the global stability of the grafcet (checking stability and execution continuous and conditional actions) and synchronizing other processes that execute partial grafcets.

**Ordinary partial grafcet:** It is a partial grafcet of which related function has a structure presented in the Annex A.3. This function will also be executed by a process.

**Expansion:** It is a partial grafcet of which related function has a structure presented in Annex A.1.
**Enclosure:** Enclosure is a partial grafcet of which related function has a structure presented in Annex A.2.

In the cases of expansion and enclosure, the functions called in the algorithms are methods related to the processes.

### 3.7. Main function

The content of the main function that will be executed is detailed in Annex A.4. VariableOrder is actually linked individually to each of the created processes other than the main one. In other words, these variables help to synchronize processes. We also preferred to consider some variables as global (but with no mutex problem) to allow easy access to these variables for all processes. These include tables of steps and transitions.
In addition, we have created a process to update information about the transition conditions. The function that will be executed by this process has a structure that looks like Annex A.5.
Fig. 10 in Annex B represent the class diagram of C++ codes for mbed platform obtained after translations.

## 4. Case study

In this section, we present a small application: a day/night sensor (by using an LDR). The display of current information was provided by a terminal using the serial port of the mbed card. It serves as visualization interface between the module and the user. Fig. 6 and 7 represent an overall view of the grafcet realized under Jgrafchart for this application.

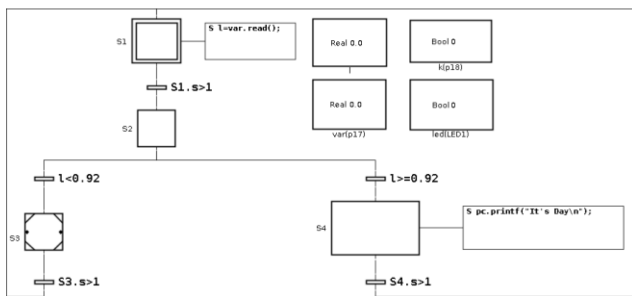



*Figure 6: Grafcet application*

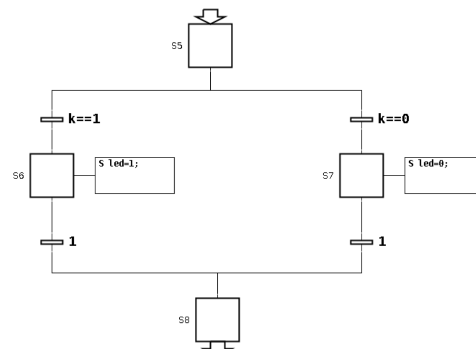



*Figure 7: Expansion of the macro-step S3*

For the step S1 at Fig. 6, we acquire the information through the Port p17 of the microcontroller LPC1768. After this, we check the value of the information in comparison with a threshold value that has been obtained by experimentation. And finally, we take a decision at steps S3 or S4 according to the value of information before restarting the process. We should notice that at the steps S1, S3 and S4, a timing of 1 second is done, but this syntax is specific to Jgrafchart.
The macro-step S3 specifies how an LED can be commanded according to the state of the sensor.
Finally, the C++ code for mbed platform obtained for this application counts less than 600 lines, and the resulting .bin file size for LPC1768 is 70.2 kB. Fig. 8 and 9 respectively show the circuit of the application and the terminal.

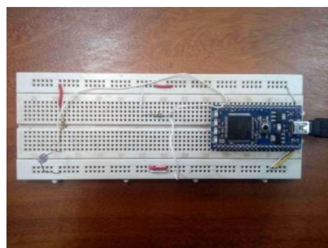


*Figure 8. Board Circuit of the application*

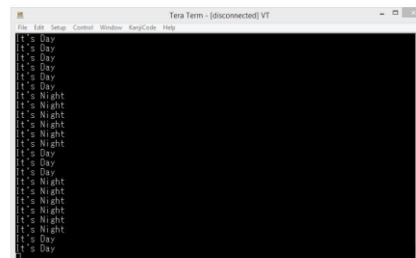



*Figure 9: Display terminal*

## 5. Conclusion

This article presents a tool for the translation of grafcets into compilable C++ code, especially for the mbed platform. We first looked for a way to represent a maximum number of elements of grafcets with the editing software JGrafchart which is just partially a grafcet editor. Subsequently, from the JGrafchart XML file of the grafcet and through a java program, we implement a method of translating the chart into a C++ code, compilable under mbed platform.

Although using GRAFCET charts (or any automata-based method) as an implementation tool increases the code size of an application, the advantages are to greatly reduce the programming and debugging stage, and to increase flexibility in the application design, without really impairing the system performance.

There are many perspectives. Firstly, translation of GRAFCET elements that we have not taken into account in this work such as forcing, actions on events, action on the clearing shoud be considered. It is also necessary to think about a development of a software which will fully support the GRAFCET standard. This software will allow editing of a GRAFCET chart (similarily to the project of S. G. Crespo [13] seeming largely unfinished) as well as the automatic generation of C++ code for the embedded platforms. A simulation of the application (like with JGrafchart) before code generation will also be a functionality. Secondly, the adaptation to others platforms (such as Arduino or FreeRTOS) other than *mbed* may be considered easily. Finally, the formal consistency of the proposed algorithms, based on a formal grammar of the grafcets to define, should be provided.

## References

[1]. E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. Model checking. MIT Press, Cambridge, MA, USA, 1999.

[2]. M. Samek. Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems. Newnes, Newton, MA, USA, 2nd edition, 2008.

[3]. M. Comlan, D. Delfieu, M. Sogbohossou, and A. Vianou. Embedding Time Petri nets. International Conference on Control, Decision and Information Technologies (CoDIT'17), Barcelona, Spain, 2017.

[4]. IEC 60848. Grafcet specification language for sequential function charts. Technical report, International Electrotechnical Commission, 2013.

[5]. IEC 61131-3. Programmable controllers - part 3: Programming languages. Technical report, International Electrotechnical Commission, 2013.

[6]. R. Julius, M. Schürenberg, F. Schumacher, and A. Fay. Transformation of grafcet to PLC code including hierarchical structures. Control Engineering Practice, 64:173 – 194, 2017.

[7]. M. Sogbohossou and A. Vianou. Formal Modeling of Grafcets with Time Petri Nets. IEEE Transactions on Control Systems Technology, 2015.

[8]. M. Sogbohossou, R. Yehouessi, T. Djara, T. Aballo, and A. Vianou. SE-LTL Model-checking on Timed GRAFCETS via ε-TPN. Current Journal of Applied Science and Technology, 38(6), 1-12, Dec. 2019.

[9]. O. Bayo, J. Rafecas, O. Gomis-Bellmunt, and J. Bergas. A grafcet-compiler methodology for c-programmed microcontrollers. Assembly automation, 28(1):55–60, Jan 2008.

[10]. mbed. https://www.mbed.com

[11]. JGrafchart, http://www.control.lth.se/Research/tools/grafchart.html

[12]. C. Johnsson and K.-E. Årzén. GRAFCHART and GRAFCET: A comparison between two graphical languages aimed for sequential control applications, Lund, 1999.

[13]. S. G. Crespo. Graphic GRAFCET diagram editor based on GeCé, Master in Information Technology-MTI, Universitat Politècnica de Catalunya, 2011

## A. The different algorithms

### A.1. Expansion

Algorithm 2 executes the grafcet of an expansion

```
1  while True do
2      If Macro-step associated is activated and following transitions are not validated then
3          Activate the Enter Step of the expansion; Execute Actions on
           Variable Order⇐=False;
4          While Test End Expansion returns "False" do if
               Variable Order has a value "True" then
5
                   If Variable Stability has a value "False" then
6
                       Call ComputeTransition;
7
                       If Exists Fireable Transitions then Call
8                          Compute Next Situation; for Each New
                           Activated Step do
9
                               Call Actions on Activation;
10
                           end
11                 Variable Order⇐=False;
               end
12         end
13     end
```

**Algorithm 2.** *Expansion Algorithm*

### A.2. Enclosure

Algorithm 3 executes the grafcet of an enclosure.

```
1  while True do
2      If Enclosing Step associated is activated then
3          Call Initialization of Situation;
4          For Each New Activated Step do
           Variab le Order⇐= False;
5          whileEnclosing Step associated is activated do if Variable
               Order has a value "True" then
6
                   if Variable Stability has a value "False" then
7
                       Call ComputeTransition;
8
                       If Exists Fireable Transitions then Call
9                          ComputeNextSituation; for Each New
                           Activated Step do
10
                               Call Actions onActivation;
11
                           end
12                 Variable Order⇐= False;
               end
13         end
14     end
15
```

**Algorithm 3.** *Enclosure Algorithm*

### A.3. Ordinary partial grafcet

Algorithm 4 executes an ordinary partial grafcet (i.e. which is not an expansion or an enclosure).

```
1 while True do
2     Call Initialization of Situation;
3     for Each New Activated Step do
4         Call Actions on Activation;
      whileTrue do
5
          if Variable Order has a value "True" then
6             if Variable Stability has a value "False" then
7                 Call ComputeTransition;
8                 if Exists Fireable Transitions then Call
9                     ComputeNextSituation; for Each New
                      Activated Stepdo
10                        Call Actions onActivation;
11                    end
12                Variable Order⇐= False
              end
13    end
14
```

**Algorithm 4.** *Ordinary partial grafcet algorithm*

### A.4. Main process

Algorithm 5 executes the main thread which controls all the threads executing the partial grafcets.

```
1 Function Test Stability
2 Variable VariableStability: Boolean;
3 If There is no Fireable Transition then
4     Variable Stability⇐=True;
5 else
6     Variable Stability⇐=False;
7 end
```

```
1 Create and Start all process;
2 while True do
3     while All variables Variable Orderd on ’thaveavalue "False" do
4         Do Nothing;
      end
5     Call Test Stability() if Variable Stability has a value "True" then
6         Execute Continue and Conditionnal Actions;
7     end
```

**Algorithm 5.** *Main Process Algorithm*

### A.5. Function to setup the value of transition conditions

Algorithm 6 is used to setup the value of the transition conditions.

```
1  while True do
2      For Each transition of the grafcet do
3          If The transition condition is timed then if The transition condition
               is Truethen
4
5                  Setup the attribute of the transition associated to the value of transition condition (as "True");
               else
6
7                  Setup the attribute of the transition associated to the value of transition condition (as"False");
               end
8
9                      If The time of activation of this step ≥ timing value then
10                         Setup the attribute of the transition associated to the value of transition condition (as "True");
11
                       else
12                         Setup the attribute of the transition associated to the value of transition condition (as "False");
13
                       end
14                 end
15             end
           end
16     end
17
```

**Algorithm 6.** *Algorithm for setting up of the value of transitionconditions*

**B. Class diagram for C++ codes for mbed platform**

Fig. 10 gives the class diagram for the mbed code to generate.

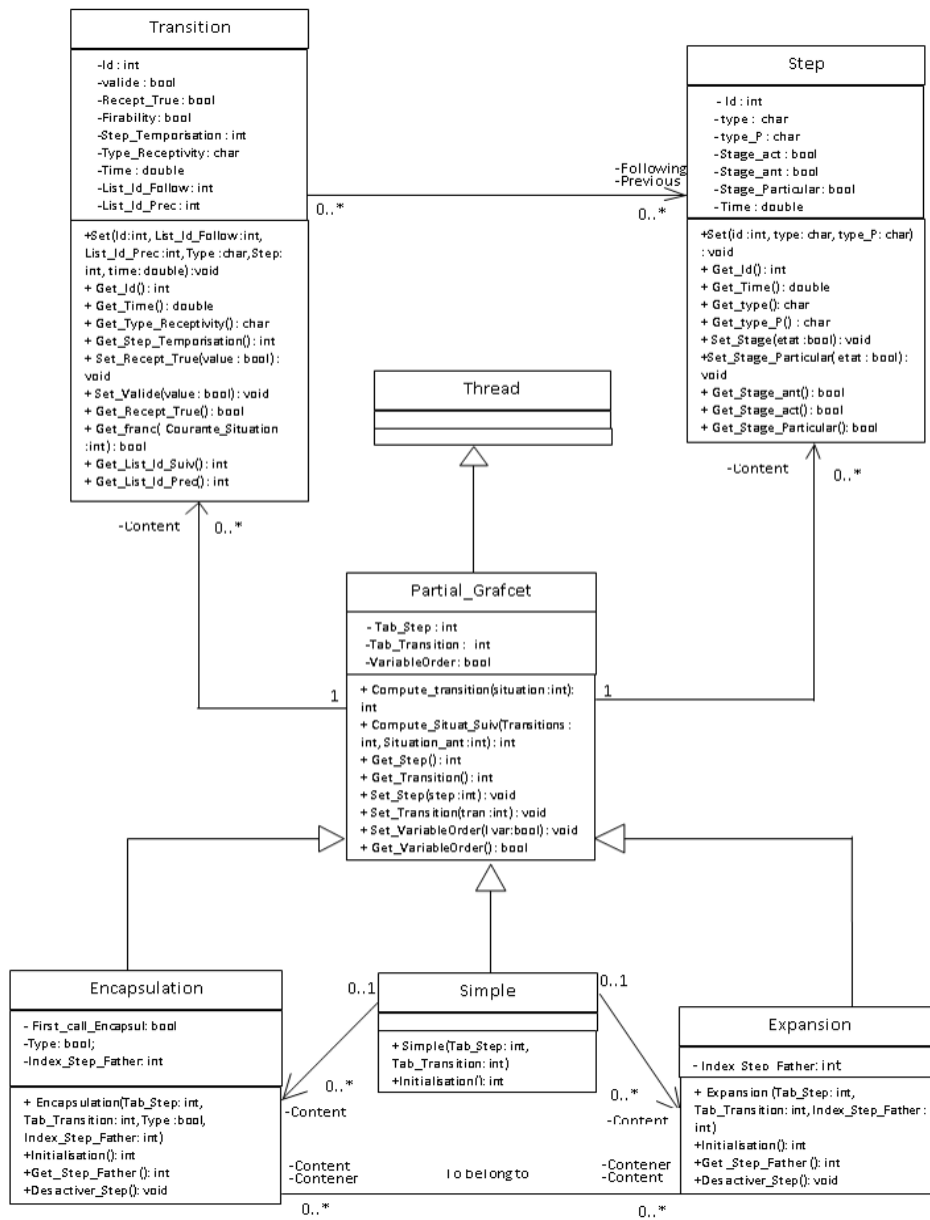


*Figure 10: Class diagram (UML)*