Model-checking on grafcets through translation into time Petri net

Médéou Sogbohossou¹ — Rodrigue Yehouessi¹ — Bernard Berthomieu²

¹Département Génie Informatique et Télécommunications
École Polytechnique d’Abomey-Calavi (EPAC), 01 BP 2009 Cotonou, BENIN
medesu.sogbohossou@epac.uac.bj, yehouessi_rodrigue@yahoo.fr

²LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, FRANCE
Bernard.Berthomieu@laas.fr

ABSTRACT. The conception of a critical automated system goes through its formal specification in order to proceed to its validation. One of the well-known formalisms to specify the behaviour of such a system is the GRAFCET standard (IEC 60848). GRAFCET being just a semi-formal language, we choose to use an intermediate language to translate without any ambiguity a grafcet model: the time Petri nets (TPN), which take into account quantitative time in a model. In this paper, we propose some verification formulas on GRAFCET charts, via the generated intermediate TPN model: CTL and SE-LTL temporal logics are used to express properties being about situations and actions of the GRAFCET chart. Then, we provide a procedure of implementation by using JGrafchart (a grafcet editor) and the model-checkers in TINA software, namely SELT (for SE-LTL properties) and MUSE (for CTL properties).

RÉSUMÉ. La conception d’un système automatisé critique passe par sa spécification formelle afin de procéder à sa validation. Un des formalismes réputés pour spécifier le comportement d’un tel système est la norme GRAFCET (IEC 60848). GRAFCET n’étant qu’un langage semi-formel, nous choisissons de passer par un langage intermédiaire vers lequel le modèle est traduit sans ambigüité: les réseaux de Petri temporels (TPN), qui prennent en compte le temps quantitatif dans un modèle. Dans cet article, nous proposons des formules de vérification sur les grafcets, via le modèle TPN intermédiaire généré: les logiques temporelles CTL et SE-LTL sont utilisées pour exprimer des propriétés portant sur les situations et les actions du diagramme grafcet. Ensuite, nous proposons une procédure de mise en œuvre passant par l’éditeur de grafcet JGrafchart et les model-checkers du logiciel TINA, à savoir SELT (pour les propriétés SE-LTL) et MUSE (pour les propriétés CTL).

KEYWORDS: grafcet (IEC 60848), Time Petri Net (TPN), model-checking, CTL, SE-LTL
MOTS-CLÉS : grafcet (IEC 60848), réseau de Petri temporel, model-checking, CTL, SE-LTL
1. Introduction

The formal verification of the automated systems [9] is essential before their realization because they are often critical systems and require important costs. There are several techniques for checking such discrete event systems: theorem proof, test, simulation and model-checking [8]. Model-checking is a computer-assisted method for the analysis of systems that can be modeled by state-transition formalisms. The model-checking software takes as input the model (an automaton) and the property to check on it. When a property is not satisfied for the studied system, model-checking may provide a counter-example.

The GRAFCET formalism [11] is widely used by the automation specialists to describe the behavior of the sequential control part of an automated system, by the means of charts in the system specification phase. This standard should not be confused with the SFC one [12, 13] intended for implementation purposes.

However, GRAFCET (and SFC) formalisms are not mathematically defined, and so they contain some ambiguities to clarify via a translation of the chart into a formal representation, such as SMV textual language [15, 1, 14], timed automata [10] or time Petri nets (TPN) [16]. TPNs are structurally (and historically) closer to the GRAFCET than the other formal models.

Thus, the novelty of the present work is to propose a procedure for doing model-checking on a GRAFCET chart (or grafcet for short) translated into time Petri net (TPN) according to [16]. Based on the state space construction of a classic TPN [3, 5], two algorithms are implemented to obtain sufficiently compact abstractions which will be the inputs of a model-checker. The first algorithm only preserves informations in the original grafcet (by abstracting extra informations appearing after the translation). The second one enhances the abstraction and displays only states where the grafcet is in a stable situation. Further, taking into account the specificities of the translation into TPN, some expressions of properties are proposed in CTL and LTL temporal logics, and concern situations and actions of the grafcet. Thanks to SE-LTL [7], it is specially possible to integrate transitions in a property formula.

For the practical experiences, the grafcet editor called JGrafchart² is used, and after implementing translation and abstractions, the model-checking is applied by means of two components of TINA software ³ [4], namely SELT (for SE-LTL model-checking) and MUSE (for CTL model-checking).

The remainder of this article is organized as follows. Section 2 shortly presents the used modeling formalisms, and introduces CTL and SE-LTL model-checking fragments. In Section 3 a set of formulas is proposed about the situations and actions of a grafcet. Section 4 describes the different practical steps to achieve model-checking of a grafcet, and contains a case study to illustrate our approach. Finally, Section 5 concludes this paper and gives some outlooks.

---

1. Acronym in French: GRÂpher Fonctionnel de Commande Etape Transition.
3. Time petri Net Analyzer (TINA), http://pauprojects.lamsi.fr/tina
2. Modeling formalisms and model-checking

2.1. GRAFCET charts

A GRAFCET chart [11] is a graphical representation modelling the behavior of the control part of an automated system. This representation consists of two parts:

- the structure describes the possible evolutions between the situations. It consists of the following basic elements: step, transition and directed link. A situation is the set of active steps at a given instant;

- the interpretation enables the relationship between the literal variables (inputs, outputs, delays, internal variables, ...) and the structure. It is done through the transition conditions (containing inputs, rising/falling edges of boolean inputs, delays, ...) and the actions (continuous actions, stored actions).

Figure 2 in Annex B shows an example of a grafcet edited with JGrafchart. It should be noticed that JGrafchart respects only partially the syntax (and the semantics) of the GRAFCET standard. For instance, a continuous action and a stored action on activation are defined respectively with qualifiers $N$ and $S$ (like in the SFC standard), and a timed variable $T_j/X_j$ on a step $i$ (with the value $T_j$ in the second unit) is denoted by $S_i.s > T_j$.

2.2. Translation of grafcet into Time Petri net

Definition 1. A Time Petri Net (TPN) [16] is a tuple $(P, T, W, W_f, W_R, \downarrow SI, \uparrow SI, M_0)$ such as:

- the nodes: $P$ is the set of places and $T$ is the set of transitions ($P \cap T = \emptyset$);
- $W : P \times T \cup T \times P \rightarrow \mathbb{N}$ defines the regular arcs between nodes (and their weights);
- $W_R : P \times T \rightarrow \mathbb{N}$ defines the read arcs;
- $W_f : P \times T \rightarrow \mathbb{N}^+ \cup \{\infty\}$ defines the inhibitor arcs;
- $\downarrow SI : T \rightarrow \mathbb{Q}^+ \ (resp. \ \uparrow SI : T \rightarrow \mathbb{Q}^+ \cup \{\infty\})$ defines the lower (resp. upper) bound of the static interval of the transitions;
- the initial marking $M_0 : P \rightarrow \mathbb{N}$.

A marking $M$ may enable some transitions in the set $T$. A transition firing is also conditioned by time information of all the enabled transitions, depending on their static intervals. A firing sequence expresses a behaviour of the modelled system. The standard semantics is used here and is more precisely recalled in a reference such as [6].

The works [16] have proposed a procedure of translating a grafcet into a TPN model, of which syntax is extended by $\epsilon$ infinitesimal delays as bounds on some transitions, allowing to simulate the synchronous semantics of GRAFCET. A extra module (called phase sequencer) is necessary to allow a transient evolution without modification of inputs as external events: it forces alternation between the reaction phase (called evolution with grafcets) and an external event production (an input change or some timed variable becomes true) in a stable situation. After adding this first module, the generation of the complete TPN is done by translating sequentially: the steps, the inputs, the timed variables, the outputs, the counter variables, the continuous and conditional actions, the stored actions and the grafcet transitions. These grafcet elements (steps, transitions, input variables, actions, ...) correspond to different but connected blocks in the resulting TPN.
The spatial complexity of the translation is polynomial with the number of nodes (steps and transitions), variables or literal terms of the grafcet.

2.3. Model-checking

A model-checking software takes as input an abstraction of the system behavior (a transition system such as a TPN state space [5]) and a property (expressed in Temporal Logic [8]) to check on the model, and answers if the abstraction satisfies or not this property. There are several types of temporal logic including: LTL (Linear Temporal Logic) to express properties on each path of the transition system and CTL (Computational Tree Logic) to express properties taking into account the branching of the different possible futures of the transition system.

A property \( p \) is formulated by means of a logical proposition (or formula), of which interpretation (i.e. true or false value) depends on a model \( M \) on which this property is expressed. Thus, the property \( p \) verified for the model \( M \) is denoted: \( M \models p \). For temporal properties about a discrete event system, the commonly used model is called labeled Kripke structure (LKS): it is a kind of state graph of which each state is labeled with some atomic propositions (in a set \( AP \) true in this state; a transition between two states is labeled by a subset \( A \) of events in \( \Sigma \). In our context, the model \( M \) is the state class graph (SCG) [4] obtained from the translation of a grafcet into an equivalent TPN [16], and the propositions concerns the marking of the different places in the TPN. Here, CTL and LTL temporal logics are used to express properties about situations and actions of the GRAFCET chart.

A path \( \pi = (s_0, A_0, s_1, A_1, s_2, A_2, ...) \) of a LKS is an alternating infinite sequence of states \((s_0, s_1, ... \) with \( s_0 \) the initial state) and events \((A_0, A_1, ... \) with \( A_i \) a set of TPN firings from the state \( s_i \)). Notation \( \pi^i \) stands for the suffix of \( \pi \) starting in the state \( s_i \).

The syntax of SE-LTL (State-Event LTL [7]) path formula is given by (where \( p \) ranges over \( AP \) and \( \alpha \) ranges over \( \Sigma \)):

\[
\phi := p \mid \alpha \mid \lnot \phi \mid \phi \lor \psi \mid \phi \land \psi \mid X \phi \mid F \phi \mid G \phi \mid \psi U \phi
\]

For the SE-LTL semantics, a path-satisfaction of formulas is defined inductively as follows (\( \mathcal{C}(s_0) \) is a subset of \( AP \) labeling \( s_0 \)):

1. \( \pi \models p \) iff \( p \in \mathcal{C}(s_0) \), and \( \pi \models \alpha \) iff \( \alpha \in A_0 \).
2. \( \pi \models \lnot \phi \) iff \( \pi \not\models \phi \),
3. \( \pi \models \phi_1 \lor \phi_2 \) iff \( \pi \models \phi_1 \) or \( \pi \models \phi_2 \),
4. \( \pi \models \phi_1 \land \phi_2 \) iff \( \pi \models \phi_1 \) and \( \pi \models \phi_2 \),
5. \( \pi \models X \phi \) iff \( \pi^1 \models \phi \),
6. \( \pi \models F \phi \) iff \( \exists k \geq 0 \) s.t. \( \pi^k \models \phi \),
7. \( \pi \models G \phi \) iff \( \forall k \geq 0 \), \( \pi^k \models \phi \),
8. \( \pi \models \phi_1 U \phi_2 \) iff \( \exists k \geq 0 \) s.t. \( \pi^k \models \phi_2 \) and \( \forall 0 \leq j < k, \pi^j \models \phi_1 \).

LTL is the restriction of SE-LTL without labels on transitions (i.e. just State LTL). Here, CTL does not consider events, like simple LTL.

The syntax of CTL state formula is given by (\( \phi \) is a path sub-formula):

\[
\phi := p \mid \lnot \phi \mid \phi \lor \psi \mid \phi \land \psi \mid E \phi \mid A \phi
\]

\[
\phi := X \phi \mid F \phi \mid G \phi \mid \phi U \phi
\]

For the CTL semantics, a state-satisfaction of formulas is defined inductively as follows:
3. Model-checking on grafcets

We distinguish the properties according to the objects of the grafcet that they handle: situations or actions.

Some properties depend on the specificities of the approach of translation proposed in [16]. Some elements in the resulting TPN report the evolution states and the stability states of a grafcet: it is the case of the places called Stable and Evolution, and the transition Evolution_end.

3.1. Properties on the structural aspect

Let $S$ be the set of steps of the considered grafcet.

The LTL properties on the structural aspect are the followings:

1) To find out whether the step $X_i$ is permanently active: $\text{G} \ X_i$
2) To test the existence of a step $X_i$ permanently active: $\bigvee_{X_i \in S} \text{G} \ X_i$
3) To verify that a step $X_i$ is never active: $\text{G} \neg X_i$

The CTL properties about the structural aspect are the followings:

1) To find out whether the step $X_i$ is permanently active: $\text{AG} \ X_i$
2) To test the existence of a step $X_i$ permanently active: $\bigvee_{X_i \in S} \text{AG} \ X_i$
3) To verify that a step $X_i$ is never active: $\text{AG} \neg X_i$
4) To know if in the future the step $X_i$ could be permanently active: $\text{EF} \text{EG} \ X_i$
5) To test the existence of any step active permanently in the future: $\bigvee_{X_i \in E} \text{EF} \text{EG} \ X_i$
6) To check whether the activity of the step $X_j$ is reachable since the one of the step $X_i$: $\text{AG} (X_i \Rightarrow \text{AF} \ X_j)$
7) To check whether active step $X_k$ is reachable from active step $X_i$ through the active step $X_j$: $\text{EF} (X_i \Rightarrow \text{EF} (X_j \land (X_j \Rightarrow \text{EF} \ X_k)))$

8) To check that it is possible to find a grafcet execution where the steps $X_i$, ..., $X_n$ are activated simultaneously: $\text{EF} (\neg \neg X_i \land \ldots \land \neg \neg X_n) \land \text{EX} (X_i \land \ldots \land \neg X_n)$
9) To check whether it is possible to return to step $X_i$ or to verify that a step $X_i$ is accessible from all grafcet situations: $\text{AG} \text{EF} \ X_i$

---

4. $\phi \Rightarrow \varphi$ is equivalent to $\neg \phi \lor \varphi$. 
10) To check if there is a graftocet situation where there is a deadlock (that is to say a situation that can no longer be left): \( \text{EF EG} (\text{Evolution} \Rightarrow \text{Evolution}_{\text{end}}) \)

11) To check if there may be total instability in the system: 
\( \text{EF EG} \neg \text{Evolution}_{\text{end}} \)

In fact, the two last properties are not valid in CTL since \( \text{Evolution}_{\text{end}} \) is an event. To make such properties valid in a State-Event CTL such as UCTL [2], any classical proposition \( Prop \) only made with events (i.e. transition firings) should be replaced by \( AX_{Prop} \text{ true} \); so, \( \text{Evolution}_{\text{end}} \) will become here \( AX_{\text{Evolution}_{\text{end}}} \text{ true} \).

3.2. Properties on the actions

Let \( S_j \) be the set of steps associated with the action \( \text{action}_j, T_{j_1} \) (resp. \( T_{j_2} \)) the set of succeeding (resp. preceding) transitions of the steps associated with the action \( \text{action}_j \).

The possible forms of \( \text{action}_j \) are:

- \( \text{action}_j \) is a continuous action: \( \bigvee_{X_i \in S_j} X_i \wedge \text{Stable} \)
- \( \text{action}_j \) is a conditioned action by \( \text{condition}_j \) (a logical expression): \( \bigvee_{X_i \in S_j} X_i \wedge \text{Stable} \wedge \text{condition}_j \)
- \( \text{action}_j \) is a stored action (translatable into \( AX_{\text{action}_j} \text{ true} \) in UCTL):
  - on activation: \( \bigvee_{t_r \in T_{j_2}} t_r \)
  - on deactivation: \( \bigvee_{t_r \in T_{j_3}} t_r \)

These different forms are used to check the following LTL and CTL properties:

1) LTL property: to show that an action \( \text{action}_1 \) always follows an action \( \text{action}_2 \):
\( \text{action}_2 \Rightarrow F \text{ action}_1 \)

2) CTL properties:
   a) To show that an action \( \text{action}_1 \) always follows an action \( \text{action}_2 \): \( AG (\text{action}_2 \Rightarrow AF \text{ action}_1) \)
   b) To show that an action \( \text{action}_1 \) is launched simultaneously with an action \( \text{action}_2 \):
\( \text{EF} (\neg \text{action}_1 \wedge \neg \text{action}_2) \Rightarrow EX (\text{action}_1 \wedge \text{action}_2) \)

Naturally, some more general property may mix up both action and step propositions.

4. Implementation of the model-checking

4.1. Procedure

To make model-checking on the graftocet, we proceed as follows:

1) The graftocet to be verified is edited under the JGrafchart software (as shown the figure 2 of the case study in Appendix B). This software generates an XML file containing information on the elements of the graftocet;

2) From the XML file, our Java program generates a .net extension file containing information about the elements of the TPN equivalent to the edited graftocet;

3) The implementation of the algorithms 1 and 2 (Annex A) allows us to obtain respectively from the file .net, a file .aut containing the information on the elements of
the automaton (with unstable and stable states) of the grafzet and another one containing only the information on the stable states of the grafzet (by disregarding unstable states);

4) The TINA ktzio tool takes the .aut file as input to generate the Kripke structure (.ktz extension file on which the verifications are made);

5) Finally, the tools SELT and MUSE (examples in Appendix C) of TINA are used to check the LTL and CTL properties on the grafzet from the Kripke structure.

4.2. Application

The illustration is based on the grafzet as shown in Figure 2 (Annex B). This grafzet models two traffic lights located respectively on a track A and a track B. It contains a transient mode (orange lights blink three times) and a steady state. From the JGrafchart XML file, we generated the equivalent TPN and the two automata of figures 3 and 4 (in Annex B, edited from the generated .aut files).

The following examples of properties are checked on the grafzet.

Verification of a LTL property with SELT tool:

- The system leaves the transient mode (firing of transition 13): TRUE. The result of this verification\(^5\) is shown in Figure 1.

![](image)

\textbf{Figure 1. Verification of the exit from the transient mode on the first automaton (with stable and unstable states).}

Verification of some CTL properties with MUSE tool:

- The street A light can stay permanently green: FALSE. Cf. Figure 5 (Annex C).
- The counter that allows blinking of the orange lights in the transient mode can reach the value 4: FALSE. Cf. Figure 6 (Annex C).
- Lights can become green or orange, simultaneously for streets A and B: FALSE. Cf. Figure 7.
- The same lights can pass simultaneously to two different colors (green and red for example): FALSE. Cf. Figure 8.

5. Conclusion

Through these works, we have shown the possibility to check properties (SE-LTL and CTL respectively with the tools SELT and MUSE of the software TINA) on a grafzet after translating it into an equivalent TPN, and subsequently into an automaton representing the state-space. This automaton is as compact as possible by abstracting much information in the TPN and by avoiding multiple interleavings due to the concurrent firings in the TPN.

\(^5\) With SELT, operators G, X and \(\land\) are denoted resp. [\{], (\}) and /\.,
Contrary to the grafcet translation into Timed Automata [10] or TSMV [14], TPNs do not allow model-checking on quantitative time properties with TCTL logic. To introduce timed properties, a perspective to our approach is to integrate observers into the TPN of the translation, to take into account delay events while model-checking the grafcet. Another perspective is the creation of a software implementing all steps of our approach: from editing a grafcet (in full conformity with the IEC60848 standard) until the verification phase. Finally, the extension of CTL to Action/State-Based Temporal Logic UCTL [2] will be an asset to generalize the expression of some properties including events of firing.

6. References


A. Algorithms

Two algorithms are proposed and implemented (in Java) to obtain sufficiently compact abstractions.

Algorithm 1. SCG (LTL): first abstraction
1. Save and stack (LIFO) the initial state;
2. while (the Stack is not empty) do
3. Unstack a state (or state class);
4. if (a grafnet transition is fireable) then
5. Fire all simultaneously fireable grafnet transitions;
6. Fire all fireable transitions for modifying literals;
7. if (the last reached state is new) then Save and stack it;
8. else if (Evolution_End is fireable) then
9. Fire transition Evolution_End;
10. Fire all fireable transitions for continuous actions;
11. if (the last reached state is new) then Save and stack it;
12. else if (Change_input or a delay transition of some timed variable model are fireable) then
13. for each fireable transition do
14. Fire all fireable transitions until a grafnet transition or Evolution_End is fireable;
15. if (the last reached state is new) then Save and stack it;
16. end
17. end
18. end

Algorithm 2. SCG (LTL): second abstraction
1. Stack (LIFO) the initial state;
2. while (the Stack is not empty) do
3. Unstack a state (or state class);
4. if (a grafnet transition is fireable) then
5. Fire all simultaneously fireable grafnet transitions;
6. Fire all fireable transitions for modifying literals;
7. if (the last reached state is new) then Stack it;
8. else if (Evolution_End is fireable) then
9. Fire transition Evolution_End;
10. Fire all fireable transitions for continuous actions;
11. if (the last reached state is new) then Stack it;
12. else if (Change_input or a delay transition of some timed variable model are fireable) then
13. for each fireable transition do
14. Fire all fireable transitions until a grafnet transition or Evolution_End is fireable;
15. if (the last reached state is new) then Save and stack it;
16. end
17. end
18. end
Algorithm 1 only preserves informations in the original grafet, by abstracting extra informations appearing after the translation: for instance, firings related to the synchronous updatings (step states and literal variables) are abstracted.

Algorithm 2 is the same as Algorithm 1, except that only state classes corresponding to the stable situations of the grafet (line 15) are saved, and only Change_input and delay transition firings from these classes are displayed.

The two algorithms assume that all possible interleavings of firings which symbolize a grafet evolution between two stable situations lead to the same global state. This assumption (and other prerequisites) were discussed in [16].

B. The case study

Figure 2. Case study

From the JGrafchart XML file of Figure 2, we have generated the equivalent TPN and the two automata of figures 3 and 4 (edited graphically with the tool ND of TINA taking as input the generated .aut files). To summarize:

- the TPN obtained is made of 75 places, 96 transitions, 205 regular arcs, 110 read arcs and 20 inhibitor arcs;
- the first abstraction (figure 3) is made of 42 states and 42 transitions;
– the second abstraction (figure 4) is made of 12 states and 12 transitions.

Figure 3. Automaton based on algorithm 1 (stable and unstable states of the grafset)

Figure 4. Automaton based on algorithm 2 (stable states of the grafset)
C. Some TiNA results of the case study

These results concern CTL properties tested with MUSE tool, only by using the second automaton (with only stable states).

Figure 5. Checking the permanent activation of the green light of the street A

Figure 6. Checking the state of the counter

Figure 7. Checking a safety property

Figure 8. Checking no two lights on at the same place

Fig. 5 shows that a green light will never stay indefinitely turned on. Fig. 6 shows that the orange lights in the transient mode will not blink more than three times. Fig. 7 displays that the lights for the two crossing streets will never allow all road users to pass through simultaneously. Finally, Fig. 8 shows that two lights may not turn on simultaneously for some user.
Adjustment module
to give auto-adaptiveness behavior to flood forecasting systems

Joel TANZOUAK*, Idrissa SARR†, Blaise YENKE*, Ndiouma BAME†, Serigne FAYE†

*The University of Ngaoundere (Cameroon),
†The University of Cheikh Anta Diop (Senegal)

RÉSUMÉ. La prévision est aujourd'hui un facteur clé dans la minimisation des dégâts causés par les inondations. En effet, les systèmes de prévision d’inondations (FFS) fonctionnent pour la plupart dans les pays développés et utilisent des modèles hydrauliques pour fournir des prévisions du niveau et/ou du débit des rivières en se basant sur les prévisions météo (NWP). Ces données de prévision sont utilisées pour fournir des alertes d’inondations; Il est donc important d’utiliser de bons modèles hydrauliques pour obtenir des données précises. De nombreux modèles hydrauliques ont été construits pour les FFS. Cependant, la différence entre les paramètres environnementaux et climatologiques entre les régions rend très difficile l’utilisation de ces FFS dans d’autres régions. De plus, l’évolution constante au cours du temps de l’environnement, causée par des facteurs anthropiques, nécessite un processus de recalage fréquent des modèles hydrauliques pour qu’ils s’alignent aux changements environnementaux. Par conséquent, il est nécessaire de construire des FFS qui s’adaptent dynamiquement aux changements environnementaux sans processus de recalage. L’objectif de cet article est de proposer une extension des FFS en introduisant un module d’ajustement qui utilise des données collectées en temps réel à partir de réseaux de capteurs combinés avec des données prévisionnelles issues des modèles hydrauliques, pour donner une capacité d’auto-adaptation dynamique aux FFS. Les résultats obtenus à partir d’expériences empiriques montrent les avantages de notre mécanisme d’ajustement dans l’auto-adaptation des FFS.

ABSTRACT. Forecasting is now a key factor in minimizing the damages caused by Flood. Indeed, Flood forecasting systems (FFS) operate mostly in developed countries and use hydraulic models to provide forecasts of river levels and/ or flow, based on numeric weather predictions (NWP). These forecast data are used to provide flood alerts, so it is therefore important to use good hydraulic models to obtain accurate flood forecast. Many hydraulic models have been built for FFS. However, the difference between environmental and climatological parameters between regions makes very difficult the use of these FFS in other regions. Moreover, the constant evolution over time of the environment, caused by anthropic factors, need a frequent process updates of hydraulic models so that they can be adapted to environmental changes. Therefore, it is necessary to build FFS that dynamically adapt to environmental changes without a recall process. The purpose of this article is to propose an extension of FFS by introducing an adjustment module that uses real-time data collected from sensor networks combined with predictive data from hydraulic models, to provide FFS with a dynamic self-adaptation ability. The results obtained from empirical experiments show the advantages of our adjustment mechanism in the self-adaptation of FFS.

MOTS-CLEFS : Modèle hydraulique, Réseau de Capteurs, Module d’ajustement, Auto-adaptabilité
KEYWORDS : Hydraulic model, Sensors network, Adjustment module, Auto-adaptiveness