

Towards Validating a Platoon of Cristal Vehicles using CSP||B^{*}

Samuel Colin¹, Arnaud Lanoix¹, Olga Kouchnarenko², and Jeanine Souquière¹

¹ LORIA – DEDALE Team – Campus scientifique
F-54506 Vandoeuvre-Lès-Nancy

{firstname.lastname}@loria.fr

² LIFC – TFC Team – 16 route de Gray
F-25030 Besançon

{firstname.lastname}@lifc.univ-fcomte.fr

Abstract. The so-called platooning problem consists in making autonomous vehicles move in a convoy. It crosses several domains: distributed systems, embedded systems, multi-agent systems and critical systems. We thus propose to use the combination named CSP||B of two well-known formal methods to assess and verify properties of this complex system. To that end we make use of previous theoretical results on CSP||B. We also illustrate how this methodology spans the multiple composition levels of the resulting model.

Key words: formal methods, CSP||B, distributed systems, case study, platooning

1 Introduction

This paper is dedicated to the validation of land transportation systems. These systems, which are both distributed and embedded, require the expression of functional as well as non functional-properties, for example time-constrained response and availability of required services. Their dual nature is problematic: distributedness may exhibit behaviours hard to understand while embeddedness imposes the satisfaction of safety/security/confidence requirements.

To address this problem we use the CSP||B combination [1] of well-established formal methods, CSP [2] and B [3]. Our case study is a convoy of so-called Cristal vehicles seen as a multi-agent system which evolves following the Influence/Reaction model (I/R) [4] in which agents are described separately from the environment.

This convoy, called a *platoon*, is a set of autonomous vehicles which have to move following the path of the leader in a row. Its control concerns both a longitudinal control, i.e. maintaining an *ideal* distance between each vehicle, and a lateral control, i.e. each vehicle should follow the track of its predecessor. As both controls can be studied independently [5] we will only focus on the longitudinal one. The Cristal driving system perceives information about its environment before producing an instantaneous

^{*} This work is supported by the French National Research Agency ANR-06-SETI-017 TACOS project, (<http://tacos.loria.fr>), and the pôle de compétitivité Alsace/Franche-Comté/CRISTAL project (<http://www.projet-cristal.net>).

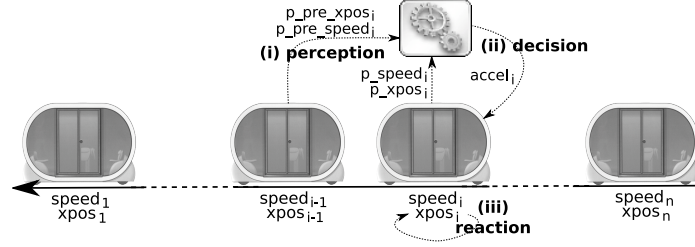


Fig. 1. A platoon of Crystals

acceleration passed to its engine. As we consider only longitudinal control, we represent the position of the i^{th} Cristal by a single variable $xpos_i$ and its velocity by $speed_i$. The behaviour of the Cristal controllers can be summarised as follows, see Fig. 1:

- (i) **perception step:** each Cristal driving system receives its velocity p_speed_i and its position p_xpos_i , from the physical part of the Cristal. Furthermore, it receives by network communication the velocity $p_pre_speed_i$ and the position $p_pre_xpos_i$ of its leading Cristal;
- (ii) **decision step:** each Cristal driving system can influence its speed and position by computing and sending to its engine an instantaneous acceleration $accel_i$. The acceleration can be negative, corresponding to the braking of the Cristal;
- (iii) **reaction step:** $xpos_i$ and $speed_i$ are updated, depending on the current speed $speed_i$ of the Cristal and a decided instantaneous acceleration $accel_i$ of the engine.

Our approach is “bottom-up”-oriented: B machines describe the various components of a Cristal vehicle while CSP expresses their assembly at the level of a single vehicle and at the level of the whole convoy³. Our experience shows that writing and checking CSP||B specifications can help eliminate errors and ambiguities in an assembly and its communication protocols.

2 Theoretical Background on CSP||B

CSP||B is a combination of formal methods aimed at exploiting the best features of CSP and B, which happen to complement each other. Indeed, basic components are B machines interacting with the rest of the world through operation calls. The assembly is provided by CSP whose processes describe how B machines are scheduled and communicate with each other. We don't explain here the B and CSP semantics though for lack of space.

The main problem with combined specifications is *consistency*: CSP and B parts should not be contradictory. The consistency is obtained through a verification technique [6] consisting of verifying the *divergence-freedom* of a B machine-CSP process coupling, and its *deadlock-freedom*.

³ CSP||B specifications are available at <http://tacos.loria.fr/platoon.zip>.

The divergence-freedom of $(P \parallel M)$ can be deduced by using a technique based on *Control Loop Invariants* (CLI) [1]. Divergence-freedom is verified by exhibiting a predicate that holds for each possible path the process P can take. Let $BBODY_{S(p)}$ be the rewriting of the p^{th} path $S(p)$ of P into B using the translation rules of [1]. Here are the most important theorems we will use throughout this paper:

Theorem 1 ([7, Theorem 1]). *If there exists a predicate CLI such that for each $BBODY_{S(p)}$ in P , $CLI \wedge I \Rightarrow [BBODY_{S(p)}] CLI$, then $(P \parallel M)$ is divergence-free.*

The deadlock-freedom of $(P \parallel M)$ can be deduced by establishing the deadlock-freedom of the P part.

Theorem 2 ([6, Theorem 5.9]). *If P is a CSP controller for M with no blocking assertion on any machine channels of M , and P is deadlock-free in the stable failures model, then $(P \parallel M)$ is deadlock-free in the stable failures model.*

The following result is useful for establishing safety properties of controlled components. It means that the trace refinement established purely for the CSP part of a controlled component suffices to ensure the trace refinement for the overall component.

Corollary 1 ([6, Corollary 7.2]). *For any controller P and any B machine M , one has if $S \sqsubseteq_T P$ then $S \sqsubseteq_T (P \parallel M)$.*

The given results are also generalised in [6] to a collection of B machine-CSP process couples.

3 Specifying a Single Cristal

A Cristal vehicle is composed of two parts: its engine and a driving system, as depicted in Fig. 2. Each part is built upon a B machine controlled by an associated CSP process.

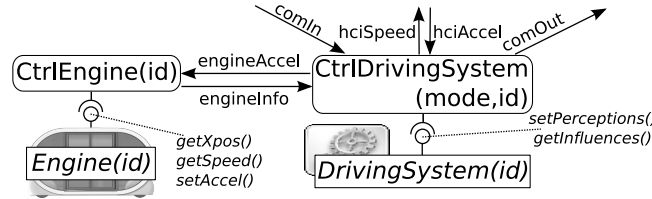


Fig. 2. Architectural view of a Cristal

The properties we want to be ensured by the model are deadlock-freedom of communications between components of the vehicle and accuracy of the information about position and speed. The former property is motivated by the fact that a vehicle could become stuck because two components wait for each other. The latter property can be interpreted as the fact that a decided acceleration should match as closely as possible the perceptions. A solution then can be to force the Cristal to alternate between “perception mode” and “reaction mode”. This is what we strive for as a safety property.

3.1 The Engine

```

MODEL Engine(id)
VARIABLES
  speed, xpos
OPERATIONS
  speed0 ← getSpeed = ...
  xpos0 ← getXpos = ...
  setAccel(accel) =
    PRE
      accel ∈ MIN_ACCEL..MAX_ACCEL
    THEN
      ANY new_speed
      WHERE new_speed = speed + accel
    THEN
      IF (new_speed > MAX_SPEED)
      THEN
        xpos := xpos + MAX_SPEED
        || speed := MAX_SPEED
      ELSE
        IF (new_speed < 0)
        THEN
          xpos := xpos - (speed × speed) / (2 × accel)
          || speed := 0
        ELSE
          xpos := xpos + speed + accel / 2
          || speed := new_speed
        END
      END
    END
  END

```

The engine is built upon a B machine that describes its knowledge about its current speed and position, and its reaction when passed a new instantaneous acceleration.

The CtrlEngine CSP controller alternates PerEngine and ActEngine. In PerEngine, we call through getSpeed?speed and getXpos?xpos the homonymous B methods to retrieve the speed and the position of the Cristal which are then passed on to engineInfo.id!xpos!speed. In ActEngine, a new instantaneous acceleration is received through engineAccel.id?accel and passed on through setAccel!accel to the B machine which calculates the vehicle position and speed updates w.r.t. this new acceleration.

The whole engine component is then defined as the composition, for a given id, of the Engine(id) machine and its CtrlEngine(id) controller.

```

PerEngine(id) =
  getXpos ? xpos → getSpeed ? speed → engineInfo.id ! xpos ! speed → ActEngine(id)
  □
  getSpeed ? speed → getXpos ? xpos → engineInfo.id ! xpos ! speed → ActEngine(id)
ActEngine(id) =
  engineAccel.id ? accel → setAccel ! accel → PerEngine(id)
CtrlEngine(id) = PerEngine(id)

```

Verification. The Engine(id) B machine consistency is successfully checked using B4Free. The CtrlEngine(id) controller *deadlock-freedom* (in the stable failures model) and its *divergence-freedom* are successfully checked with FDR2.

The composition of the B machine and the controller is verified for *divergence-freedom* by applying Theorem 1: it is specific to CSP||B and is not supported by tools hence the translation to B is done by hand. The chosen CLI is actually as simple as the \top predicate modulo the mandatory typing predicates. Then, by way of Theorem 1, we deduce that (CtrlEngine(id) || Engine(id)) is divergence-free. *Deadlock-freedom* of (CtrlEngine(id) || Engine(id)) is obtained from the deadlock-freedom of CtrlEngine(id) and the application of Theorem 2 as well.

3.2 The driving system

The (CtrlDrivingSystem||DrivingSystem) controller||B machine construction is built in a similar way. This driving system can update its perceptions and decide of an acceleration passed to the engine later on. hciSpeed, hciAccel correspond to the interaction

with a human driver (if the vehicle is in SINGLE or LEADER mode). `comIn` and `comOut` correspond to the interaction with the leading and following vehicle (PLATOON mode). `engineInfo` and `engineAccel` are used to exchange with the engine.

Using the same techniques and theorems as for the engine, the driving system is shown divergence-free and deadlock-free.

3.3 The Cristal(*mode*,*id*) Assembly

A Cristal is defined as the composition of the engine and the driving system:

$$\text{Cristal}(\text{mode}, \text{id}) = (\text{CtrlDrivingSystem}(\text{mode}, \text{id}) \parallel \text{DrivingSystem}(\text{id})) \parallel \{\text{engineInfo}, \text{engineAccel}\} (\text{CtrlEngine}(\text{id}) \parallel \text{Engine}(\text{id}))$$

Divergence-freeness is obtained by applying the generalised version of Theorem 1 to the divergence-freeness of both components $(\text{CtrlEngine}(\text{id}) \parallel \text{Engine}(\text{id}))$ and $(\text{CtrlDrivingSystem}(\text{mode}, \text{id}) \parallel \text{DrivingSystem}(\text{id}))$. Deadlock-freeness of the Cristal stems from deadlock-freeness of $(\text{CtrlEngine}(\text{id}) \parallel \text{CtrlDrivingSystem}(\text{mode}, \text{id}))$ and by applying the generalised version of Theorem 2.

Let us note that earlier versions of the models had deadlocks exhibited by the FDR2 tool: having access to the faulty traces helped us understand the errors and modify the driving system controller with a tighter scheduling leading to deadlock-freeness.

Safety Property. The property stating that perception and reaction should always alternate can be re-expressed as a CSP process:

$$\text{Property}(\text{id}) = \text{engineInfo.id?xpos?speed} \rightarrow \text{engineAccel.id?accel} \rightarrow \text{Property}(\text{id})$$

Checking that the Cristal meets this property is akin to checking that there is a trace refinement between it and the Cristal. This is achieved by checking that $\text{Property}(\text{id}) \sqsubseteq_T \text{CtrlEngine}(\text{id}) \parallel \text{CtrlDrivingSystem2}(\text{mode}, \text{id})$, from which it can be deduced by Corollary 1 that $\text{Property}(\text{id}) \sqsubseteq_T \text{Cristal}(\text{mode}, \text{id})$: the property is refined hence satisfied.

4 Specifying a Platoon of Cristals

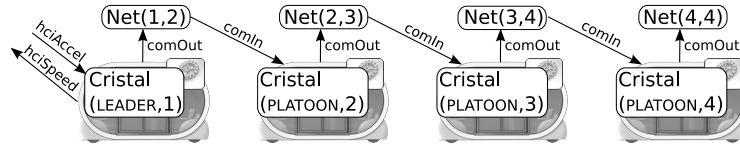


Fig. 3. A Platoon of four Cristals

Once we have a correct model for a single Cristal, we can focus on the specification of a platoon, as shown Fig. 3. We want the Cristals to avoid going stale when they are

in the PLATOON mode. This might happen because one Cristal waits for information from its leading Cristal, for instance, i.e. the communications are deadlocked.

The first Cristal of the platoon runs in the LEADER mode, while the others run in the PLATOON mode. A process $\text{Net}(\text{id}, \text{id2})$ is associated with each Cristal for managing communication: it receives information from id before sending these data to id2 . Finally, the platoon is defined by the parallel composition of all the Cristals and all the Nets, synchronised on the communication channels:

$$\text{Platoon}(n) = (\text{Cristal}(\text{LEADER}, 1) \parallel \parallel_{\text{id} \in \{2..n\}} \text{Cristal}(\text{PLATOON}, \text{id})) \parallel_{\{(\text{comIn}, \text{comOut})\}} (\parallel_{\text{id} \in \{1..n-1\}} \text{Net}(\text{id}, \text{id}+1)) \parallel \text{Net}(n, n)$$

Verification. Using FDR2, we successfully check that $\text{Net}(\text{id}, \text{id2})$ is deadlock-free and divergence-free. As each Cristal and each Net have been proved divergence-free, the platoon is divergence-free. To achieve consistency checking, the parallel composition of the CSP parts of each Cristal and Net is shown deadlock-free, thanks to FDR2. Consequently, by Theorem 2 the platoon is deadlock-free too. This verification validates that the communications (expressed through the Nets components) do not deadlock.

5 Conclusion

The development of a new type of urban vehicle and the need for its certification necessitate their formal specification and validation. We propose a formal CSP||B specification development of an autonomous vehicle's components, and an architecture for assembling vehicles in a convoy to follow the path of the leader vehicle in a row. Application of known results to the composition in the CSP||B framework and verification using existing tools – the FDR2 model-checker and the B4Free prover – allow us to ensure the consistency of the whole multi-agent system, in a compositional manner. Having formal CSP||B specifications helps – by establishing refinement relations – in preventing incompatibility among various implementations. Moreover, writing formal specifications helps in designing a way to manage the multiple architectural levels.

References

1. Treharne, H., Schneider, S.: Using a process algebra to control B OPERATIONS. In: 1st International Conference on Integrated Formal Methods, Springer Verlag (1999) 437–457
2. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985)
3. Abrial, J.R.: The B Book. Cambridge University Press (1996)
4. Ferber, J., Muller, J.P.: Influences and reaction : a model of situated multiagent systems. In: 2nd Int. Conf. on Multi-agent Systems. (1996) 72–79
5. Daviet, P., Parent, M.: Longitudinal and lateral servoing of vehicles in a platoon. In: Proceeding of the IEEE Intelligent Vehicles Symposium. (1996) 41–46
6. Schneider, S.A., Treharne, H.E.: CSP theorems for communicating B machines. Formal Aspects of Computing, Special issue of IFM'04 (2005)
7. Schneider, S., Treharne, H.: Communicating B machines. In Bert, D., Bowen, J.P., Henson, M.C., Robinson, K., eds.: Formal specification and development in Z and B (ZB 2002). Volume 2272 of LNCS., Springer Verlag (2002) 416–435