Dossier de qualification aux fonctions de maître de conférences

Section 27

Fabien Siron

Campagne de qualification 2024

Mots clés : systèmes temps-réel, langages réactifs et synchrones, méthodes formelles.

Table des matières

Remarques préliminaires

L'entreprise Krono-Safe aujourd'hui n'existe plus ; elle a été renommée Asterios Technologies à la suite de son rachat par le groupe Safran. Toutefois, dans ce document, nous utiliserons l'appellation Krono-Safe du fait que la plupart des activités mentionnées ont eu lieu essentiellement avant le rachat.

D'autre part, ma thèse de doctorat ayant été effectuée en partenariat avec une entreprise — Krono-Safe — et de par le contexte industriel de ce travail — financement CIFRE —, je n'ai eu que peu d'occasion d'enseigner.

Enfin, le lecteur trouvera en annexe le rapport de soutenance, l'attestation de réussite du doctorat, l'attestation de service d'enseignement ainsi qu'une sélection de deux articles de conférence.

1 Curriculum Vitæ

1.1 Etat civil

Nom :	Siron
Prénoms :	Fabien, Lionel
Date et Lieu de naissance :	Le 23 janvier 1996, à Clichy
Situation familiale :	Marié, père d'un enfant
Adresse Professionnelle :	Asterios Technologies 16 Avenue Carnot 91300 Massy
Adresse Personnelle :	*sur demande*
Courriels :	fabien.siron@asterios-technologies.com (professionnelle) fabien.siron@epita.fr (académique et personnelle)
Téléphone :	*sur demande*

1.2 Expérience professionnelle

Je viens de terminer ma thèse de doctorat — sous financement CIFRE — encadrée académiquement par le centre Inria de l'Université Côte d'Azur, et avec le support de l'entreprise Krono-Safe. Pendant ces trois ans, j'ai aussi eu l'occasion d'enseigner à EPITA ainsi que d'encadrer deux étudiants en stage de fin d'étude et un groupe d'étudiant en projet de fin d'étude, dans les deux cas avec le support de l'entreprise Krono-Safe. Depuis la fin de mes travaux de thèse, je me concentre sur le prototypage d'un nouveau produit proposant des techniques de vérification formelle pour le langage synchrone PsyC, développé par Krono-Safe. Mon parcours se compose essentiellement des étapes suivantes :

- **2024-maintenant** : Ingénieur Logiciel R&D, équipe *Core*, à Krono-Safe
  Prototypage d'un nouveau produit de vérification formelle pour PsyC, basé sur les résultats de mes travaux de thèse.
- **2022-maintenant** : Enseignant vacataire, à EPITA
  Enseignement d'un cours sur la modélisation temps-réel basée sur la technologie de Krono-Safe, ainsi que d'un module complet sur le test et la validation logicielle pour la spécialité embarqué et temps-réel d'EPITA (M1/M2).
- **2020-2023** : Ingénieur Doctorant (CIFRE), équipe-projet *Kairos*, à Inria et Krono-Safe
  Définition d'une méthodologie de vérification formelle pour le langage PsyC, développé par Krono-Safe.
- **2019-2021** : Ingénieur Logiciel R&D, équipe *Checker*, à Krono-Safe
  Participation au développement d'un outil de validation automatique des compilations effectuées par le compilateur produit par Krono-Safe dans un contexte de certification aéronautique.

— **2019** : Stage R&D, équipe *Modeling*, à Krono-Safe
Mise en place d'une solution de génération de code incrémentale basée sur des approches dites de *Conception Dirigée par les Modèles*.

— **2017** : Stage R&D, service *ThereSIS*, à Thales Research & Technologies
Mise en place d'une plateforme à criticité mixte — mêlant applications de vision télécom — basée sur de la virtualisation.

— **2016-2017** : Assistant de travaux pratiques, à EPITA
Encadrement et correction de travaux pratiques en programmation d'étudiants du cycle préparatoire d'EPITA.

— **2016** : Stage de recherche, *Laboratoire Système et Sécurité (LSE)*, à EPITA
Développement initial du support du protocole *Netlink* dans le traceur système de *Linux* — *strace* — via le programme (et financement) *Google Summer of Code*.

1.3 Diplômes

2023 : **Diplôme de Doctorat, Informatique**
Université Côte d'Azur, Laboratoire I3S/Kairos, Centre Inria d'Université Côte d'Azur
Soutenance effectuée le 11 décembre au centre Inria de Paris
Sujet de thèse : *"Méthodologie de vérification formelle de propriétés temporelles pour les applications temps-réel critiques basées sur le concept de temps logique "*
Direction de thèse :

Dumitru Potop-Butucaru,	Chargé de Recherche à l'Inria,	Directeur
Robert de Simone,	Directeur de Recherche à l'Inria,	Co-Directeur
Damien Chabrol,	Responsable Innovation à Krono-Safe,	Encadrant
Amira Methni,	Architecte Logiciel à Krono-Safe,	Encadrante

Jury :

Pierre-Loïc Garoche,	Professeur à l'ENAC,	Rapporteur (Président)
Reinhard von Hanxleden,	Professeur à l'université de Kiel,	Rapporteur
Timothy Bourke,	Chargé de Recherche à l'Inria,	Examinateur

2019 : **Diplôme d'Ingénieur, Informatique**
Ecole pour l'Informatique et les Techniques Avancées (EPITA), au Kremlin-Bicêtre
Majeure : Génie Informatique en Systèmes Temps-Réel et Embarqués (GISTRE)
Projet de Fin d'Etude : *"Simulation de circuit accéléré sur processeur graphique"*
Stage de Fin d'Etude : *"Génération incrémentale de modèle temps-réel"* à Krono-Safe

2014 : **Diplôme de Baccalauréat Scientifique, Informatique et Sciences du Numérique**
Lycée Claude Nicolas Ledoux, à Besançon
Mention : Bien

1.4 Langues

Français : Natif
Anglais : Lu, écrit et parlé (TOEIC 900)

1.5 Compétences techniques

Systèmes d'exploitation :	Linux, Windows
Langages Informatiques :	OCaml, Python, C, C++, VHDL, Verilog, PsyC, Esterel, Lustre, Scade
Compétences :	Systèmes Embarqués, Systèmes Temps-Réel, Méthodes Formelles, Virtualisation, Sémantiques de Langage, Compilation, Model-Checking

2 Activités d'enseignement

2.1 Enseignement

J'ai effectué la totalité de mes enseignements à l'Ecole Pour l'Informatique et les Techniques Avancées (EPITA), une école d'ingénieur du groupe IONIS spécialisée en informatique. La première partie de l'enseignement s'est faite au cours de mes études en tant qu'assistant enseignant de travaux pratiques (2016-2017), et la deuxième partie s'est faite plus récemment, dans le cadre de ma thèse, en tant qu'enseignant vacataire (2022-2023).

J'ai enseigné à un niveau L2 (cycle préparatoire intégré, deuxième année), M1 et M2 (cycle ingénieur, spécialisation) :

— L'enseignement L2 consistait essentiellement à de l'encadrement de travaux pratiques de programmation pour environ 40 étudiants.

— L'enseignement M1/M2 consistait (et consiste encore) à de l'enseignement de cours (CM), travaux dirigés (TD) et travaux pratiques (TP), en spécialisation (ou majeure) temps-réel et embarqué (appelée GISTRE) pour 40 à 50 étudiants.

Le tableau ci-dessous résume ces différents enseignements.

	Intitulé	CM	TD	TP	Vol.	Année	Public
L2	Travaux Pratiques de programmation	-	-	18h	18h	2016-2017	Prepa, EPITA
M1	Conception temps-réel avec PsyC	2h	-	2h	4h	2022	GISTRE, EPITA
		2h	-	2h	4h	2023	GISTRE, EPITA
M2	Test et Validation	9h	3h	3h	15h	2022	GISTRE, EPITA
		7.5h	1.5h	3h	12h	2023	GISTRE, EPITA
	Total heures depuis 2022	20.5h	4.5h	10h	35h	2022-2023	EPITA
	Total heures	20.5h	4.5h	28h	53h	2016-2023	EPITA

Travaux Pratiques de Programmation

Pour les travaux pratiques de programmation, les sujets étaient écrits par des professeurs titulaires. Notre rôle d'assistant était d'encadrer et d'animer la session de TP ainsi que de corriger les rendus. Ces TPs visaient à initier les étudiants à la programmation en langage C et Go par la pratique (e.g., structures de données, algorithmes, programmation système . . .). Chaque séance durait 2h et était effectué en binôme.

Conception temps-réel avec PsyC

Le but de cet enseignement — à destination d'étudiants de la majeure GISTRE — était initialement de complémenter le cours de mon directeur de thèse D. Potop-Butucaru — nommé " Approche synchrone pour la conception temps-réel " — par une vision plus industrielle utilisant les outils produits par l'entreprise Krono-Safe. Pour cela, on m'a alloué une séance de 4h découpée en deux, composée de 2h de cours magistral et 2h de travaux pratiques. Je me suis occupé de la préparation du cours, du sujet de TP, de la séance ainsi que des corrections, en toute autonomie, mais sous le contrôle de D. Potop-Butucaru. Cet enseignement a commencé en 2022, a été réitéré en 2023 et le sera sans doute en 2024.

Le cours présente un aperçu des différents modèles de programmation traitant du temps ainsi qu'une description du langage PsyC et de la technologie Asterios, développés à Krono-Safe. Le TP contient ensuite une série d'exercices de modélisation temps-réel à effectuer avec le langage PsyC.

Test et validation

Du fait de la satisfaction des étudiants et de l'équipe pédagogique pour le cours précédent, on m'a proposé ensuite de faire un module entier — en partenariat avec un autre jeune enseignant vacataire — sur des sujets de test & validation logicielle. Pour cela, l'équipe pédagogique nous a donné 21h

par an et le cours ainsi que les TD/TPs ont été préparé avec le soutien et l'aide du responsable pédagogique de majeure, G. le Gourrierec. Bien qu'ayant préparé l'essentiel du cours, je me suis chargé d'enseigner 6 séances sur les 8 avec préparation des sujets de TD/TPs ainsi que des corrections (sauf les deux séances dont je ne m'occupais pas).

Le cours présente un tour des différentes techniques de test et validation, en commençant par les approches de test traditionnelles, incluant les approches fonctionnelles et structurelles, pour aller jusqu'aux techniques exhaustives, incluant l'analyse statique et le model-checking, en passant par d'autres techniques intermédiaires comme le test aléatoire (ou fuzzing).

2.2 Responsabilités pédagogiques

Au-delà des activités d'enseignement, on m'a aussi confié d'autres activités pédagogiques :

— Tout d'abord, en juillet 2023, j'ai été invité à faire partie du jury des soutenances de stage de fin d'étude de la promotion 2023, en spécialisation embarqué et temps-réel (GISTRE). Cela consistait en :

1. corriger et noter une partie des rapports de stage (13 rapports sur 26 à la session de juillet) ;
2. siéger au jury des soutenances (26 soutenances sur 3 jours pour un total de 20h) ;
3. faire des retours constructifs aux étudiants dont j'ai corrigé le rapport.

— Ensuite, en partenariat avec Krono-Safe, j'ai encadré un Projet de Fin d'Etude dont le sujet était « contrôle temps-réel d'un bras robotique » utilisant les technologies de l'entreprise mentionnée. L'encadrement consistait en un suivi du groupe toutes les deux semaines et au support des différents outils de l'entreprise, coassuré par moi-même ainsi qu'un collègue de Krono-Safe.

— J'ai aussi encadré deux étudiants en stage de fin d'étude (en 2021 et 2022), tous deux sur le sujet de « génération aléatoire de code PsyC », avec l'entreprise Krono-Safe. Cette activité consistait en un suivi hebdomadaire des activités des stagiaires ; ils ont tous deux validés leur stage, le deuxième étant recruté par l'entreprise aujourd'hui. Cette activité n'est pas directement une responsabilité pédagogique dans le contexte de l'école, car mon rôle consistait seulement en de l'encadrement industriel.

3 Activités de recherche

3.1 Contexte de la thèse

J'ai effectué ma thèse de doctorat entre 2020 et 2023, dirigé par D. Potop Butucaru et R. de Simone, au laboratoire I3S/Kairos du centre Inria de l'Université Côte d'Azur. Mes deux directeurs de thèse sont issus des communautés des langages et modèles synchrone-réactifs, et plus spécifiquement, des travaux de compilation et de vérification des langages Esterel et Lustre, langages ayant grandement inspiré l'état de l'art industriel actuel pour la construction de systèmes critiques (avionique, ferroviaire ...)

Ma thèse s'inscrit dans un partenariat entre l'entreprise Krono-Safe et l'Inria. Krono-Safe est une entreprise issue du CEA [1] industrialisant les résultat du projet de recherche Oasis sous le nom Asterios, une solution de conception et d'intégration pour le logiciel temps-réel. Cette solution est composée d'un langage synchrone-réactif appelé PsyC, et d'outils permettant de l'exploiter (compilateur, simulateur et noyau temps-réel). Ma thèse s'inscrit dans un objectif de proposition de solutions d'aide au design pour le langage PsyC en se focalisant sur les techniques de vérification formelle.

1. Commissariat à l'énergie atomique et aux énergies alternatives

3.2 Résumé de la thèse

Les systèmes temps-réel critiques doivent respecter des contraintes temporelles strictes qui doivent être considérées tout au long du cycle logiciel. Cependant, du fait que les temps d'exécution exacts ne sont généralement pas connus lors de la conception, le temps logique fournit un moyen d'abstraire les contraintes temporelles et les exécutions du temps physique, dépendant de la plateforme matérielle.

Dans cette thèse, nous nous sommes concentrés sur deux formalismes basés sur le temps logique. L'approche Synchrone-Réactive abstrait totalement le temps physique en utilisant des bases de temps discrètes sur lesquelles les calculs sont déclenchés. L'approche de Temps d'Exécution Logique utilise des bases de temps logique pour représenter non seulement les instants de déclenchement, mais aussi les durées des calculs élémentaires. Dans notre travail, nous avons commencé par définir une unification des approches Synchrone-Réactives et de Temps d'Exécution Logique, fournissant un cadre formel naturel pour définir la sémantique de PsyC, un langage temps-réel industriel expressif.

Nous avons défini deux sémantiques formelles pour celui-ci : une sémantique native à grands pas, préservant les durées logiques des intervalles de temps, définie par des règles structurelles opérationnelles ; et une sémantique synchrone à petits pas définie par traduction vers un langage Synchrone-Réactif étendant les durées des intervalles de temps à une succession de transitions atomiques. Par la suite, nous avons montré que ces deux sémantiques sont équivalentes. Cette formalisation de la sémantique de PsyC nous a permis de définir une méthodologie de vérification formelle pour PsyC basée sur du model-checking symbolique. Pour réduire l'espace d'état pendant le model checking, nous avons aussi définis une technique d'optimisation inspirée du model-checking temporisé. Enfin, nous avons spécifié les exigences temporelles que nous voulions vérifier via un langage de spécification de contrainte d'horloge — CCSL — qui ont ensuite été traduites en observateurs synchrones.

3.3 Perspectives de recherche

A l'issue de ma thèse, j'aimerais m'intéresser à plusieurs problématiques liées à la spécification, la conception et l'implémentation des systèmes temps-réel embarqués, et plus généralement, les systèmes cyber-physiques (CPS).

Tout d'abord, il serait intéressant de proposer des extensions inspirées du Temps d'Exécution Logique pour les langages Synchrone-Réactifs traditionnels tels qu'Esterel ou bien Lustre afin de faciliter leur implantation temps-réel. Ces travaux se baseraient directement sur le critère d'équivalence de ces deux formalismes — issu de mes travaux de thèse — et permettrait la définition de modèles d'exécution plus flexibles, nécessaires dans l'industrie.

Ensuite, il serait aussi intéressant d'améliorer les méthodes et techniques liés au langage de spécification de contraintes temporelles tel que CCSL. Après un retour d'expérience industriel, il semble que CCSL peut être plus adapté que les logiques temporelles (utilisées traditionnellement) pour spécifier des contraintes temporelles dans un système à échelles de temps multiples. Ma thèse propose un début d'encodage en CCSL d'exigences issues de l'industrie telles que les latences de bout-en-bout, mais il convient de compléter ces définitions pour couvrir toutes les variantes. D'autre part, il reste à explorer les différentes méthodes de synthèse ainsi que de vérification pour CCSL qui puissent s'inscrire dans une réelle méthodologie de conception de CPS.

Enfin, il serait aussi intéressant de continuer d'explorer les différentes techniques de vérification pour les langages à base de temps logique. Tout d'abord, il conviendrait d'approfondir les sujets d'optimisation considérant les durées en temps logique pour le model-checking symbolique. D'autre part, il conviendrait aussi d'explorer d'autres techniques moins considérées pour ces formalismes telles que la vérification déductive.

3.4 Publications

Mes travaux de thèse ont mené à la publication des articles suivants :

Conférences et workshops internationaux

1. **"Semantics fondations of PsyC based on synchronous Logical Execution Time".** Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni, In *TCRS 2023 - Time Centrics Reactive Software, San Antonio, Texas, U.S.A., 2023, url :https://dl.acm.org/doi/abs/10.1145/3576914.3587495* (Article Long).
2. **"The synchronous Logical Execution Time paradigm".** Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni. In *ERTS 2022 – Embedded Real Time Systems, Toulouse, France, 2022, url :https://inria.hal.science/hal-03694950/* (Article Long).
3. **"(WIP :)Programming and verifying real-time design using logical time".** Fabien Siron, Dumitru Potop-Butucaru, Robert de Simone, Damien Chabrol, and Amira Methni. In *FDL 2021 - Forum on specification & Design Languages, Antibes, France, 2021, url :https://inria.hal.science/hal-03537976/* (Article Court).

Les deux premiers articles de cette section — 1. et 2. — sont ceux sélectionnés et joints en annexe. L'article 2 présente un nouveau paradigme, intitulé le temps d'exécution logique synchrone (abrégé sLET), présenté comme une extension du paradigme de temps d'exécution logique (communément abrégé LET) avec des primitives de synchronisation sur des horloges logiques. Ce formalisme permet de fournir un socle théorique pour l'étude du langage PsyC, nécessaire pour l'étude de méthodes d'analyse et de vérification pour ce dernier. L'article 1 propose une suite au précédent en définissant un critère d'équivalence entre temps d'exécution logique synchrone et synchrone traditionnel afin de pouvoir réutiliser les méthodes et outils d'analyse des langages synchrone-réactifs traditionnels sur le langage PsyC, et par extension, sur tous les langages à base de LET.

Conférences et workshops nationaux

4. **"Vérification d'applications temps-réel basées sur le paradigme de logical execution time (LET)".** Fabien Siron, Dumitru Potop-Butucaru, Robert de Simone, Damien Chabrol, and Amira Methni. In *ETR 2021 - École d'Été Temps Réel, Poitiers, France, 2021, url :https://hal.inria.fr/hal-03545758/* (Article Court).

Rapport de recherche

5. **"Formal Semantics of the PsyC language".** Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni. In *Research report, INRIA Sophia Antipolis–Méditerranée, France, 2023, url :https://inria.hal.science/hal-04088177/*.

3.5 Présentations

Mes travaux de thèse m'ont aussi permis d'effectuer les présentations suivantes :

— Présentations des articles 1. à 4. dans les conférences correspondantes.

— **"Synchronous Logical Execution Time : towards formal verification".** à *Synchron 2021, La Rochette, France* (sans acte).

— **"Formal verification of temporal properties for real-time safety-critical applications based on logical time".** à *Synchron 2023, Kiel, Allemagne* (sans acte).

3.6 Productions Logicielles

Enfin à l'issue de ma thèse et lié à mes différentes expériences scolaires et professionnelles, j'ai produit les logiciels suivants qui sont pratiquement tous en accès libre :

Nom	Année	Langages	Description	Lien
PsykAnalyst	2023	OCaml, Lustre	Implémentation de la méthodologie proposée par ma thèse de doctorat : un outil de vérification formelle pour le langage PsyC basé sur des outils de model-checking SAT et BDD.	Pas disponible
Clock	2021	OCaml, SAT	Implémentation d'un simulateur symbolique de CCSL basé sur un solveur SAT contenant plusieurs heuristiques.	Bitbucket : https://bitbucket.org/Saruta_/clock/
Pegase	2019	Python, OpenCL	Conception d'un simulateur de circuit numérique accéléré grâce à une unité de calcul graphique (GPU). L'outil génère des clusters au niveau *gate-level* ordonnancé et synchronisé avec des barrières temporelles. Il s'agit de mon projet de fin d'étude (PFE) de mon cycle ingénieur à EPITA.	Bitbucket : https://bitbucket.org/Saruta_/gpgpu_accelerated_gl_simulator/
strace	2017	C, Linux	Implémentation du support initial de Netlink dans le traceur système du noyau Linux, strace. Ce projet a été financé par le programme Google Summer of Code (GSoC) et a été effectué lors de mon stage de fin de cycle préparatoire au laboratoire de l'école EPITA.	Github : https://github.com/strace/strace

SIRON Fabien

Année universitaire 2022-2023

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

UNIVERSITÉ CÔTE D'AZUR

DOCTORAT DE L'UNIVERSITÉ CÔTE D'AZUR
PROCÈS-VERBAL DE SOUTENANCE DE THÈSE

Nom et Prénom :	SIRON Fabien Date de naissance : 23 janvier 1996
Doctorat :	INFORMATIQUE
Titre de la thèse :	Méthodologie de vérification formelle de propriétés temporelles pour les applications temps-réel critiques basées sur le concept de temps logique
École Doctorale :	STIC - Sciences et Technologies de l'Information et de la Communication
Unité de recherche :	INRIA INRIA - Temps Logique Multiforme pour Conception de Systèmes Cyber-Physiques
Direction de la thèse :	Dumitru POTOP-BUTUCARU et Robert De simone
Lieu de soutenance :	2 Rue Simone IFF, 75012 Paris
Date et heure :	11 décembre 2023 à h00
Soutenance	☑ PUBLIQUE ☐ À HUIS-CLOS
Cotutelle de thèse :	☐ OUI ☑ NON

Président du jury * : Pierre-Loïc Garoche **(À COMPLÉTER)**

Le jury prononce:

☒ l'admission du candidat au titre de docteur de l'Université Côte d'Azur

☐ l'ajournement du candidat

A l'issue de la soutenance Monsieur SIRON Fabien a prêté serment ** **: ☐ Oui ☐ Non**

La Direction de la thèse ne signe en aucun cas le procès-verbal de soutenance.
Pour être valides, les documents de soutenance ne doivent pas être modifiés.

Civilité, NOM, Prénom	Fonction	Titre	Visio conférence	Signature
M. Pierre-Loïc GAROCHE	Rapporteur	Professeur	NON	
M. Reinhard VON HANXLEDEN	Rapporteur	Professeur	NON	
M. Timothy BOURKE	Examinateur	Chargé de recherche	NON	

* Article 18 de l'arrêté du 25 mai 2016 fixant le cadre national de la formation et les modalités conduisant à la délivrance du diplôme national de doctorat : « ***Les membres du jury désignent parmi eux un président. Le président doit être un professeur ou assimilé ou un enseignant de rang équivalent.*** »

Arrêté du 15 juin 1992 fixant la liste des fonctionnaires assimilés aux professeurs des universités pour la désignation des membres du conseil national des universités, modifié par l'arrêté du 19 février 2007 : « *Un maître de conférences HDR ou chargé de recherche HDR n'est pas assimilé professeur des universités.* »

Le président du jury est mandaté pour signer le procès-verbal de soutenance et le rapport de soutenance pour toute personne ayant une dérogation pour participation en visioconférence.

** En application de l'article L612-7 du code de l'éducation, les établissements d'enseignement supérieur et de recherche mettent en œuvre, dans le cadre des soutenances de leurs doctorantes et doctorants, une prestation de serment d'intégrité scientifique.

Article 19bis de l'arrêté du 26 août 2022 de la formation doctorale modifiant l'arrêté du 25 mai 2016 : « *A l'issue de la soutenance et en cas d'admission, le docteur prête serment, individuellement, en s'engageant à respecter les principes et exigences de l'intégrité scientifique dans la suite de sa carrière professionnelle, quel qu'en soit le secteur ou le domaine d'activité.* »

NB : Le directeur de thèse participe au jury mais ne prend pas part à la décision **et ne signe pas le procès-verbal de soutenance** (cf. article 18 de l'arrêté du 25 mai 2016).

SIRON Fabien

Année universitaire 2022-2023

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

UNIVERSITÉ CÔTE D'AZUR

DOCTORAT DE L'UNIVERSITÉ CÔTE D'AZUR
RAPPORT DE SOUTENANCE

Nom et prénom du doctorant : SIRON Fabien

Titre de la thèse : Méthodologie de vérification formelle de propriétés temporelles pour les applications temps-réel critiques basées sur le concept de temps logique

École Doctorale : STIC - Sciences et Technologies de l'Information et de la Communication

Date de la soutenance : 11 décembre 2023

Président du jury : Pierre-loïc Garoche (À COMPLÉTER)

Nombre de pages du rapport : 137

Membres du jury :

Nom	Signature	Nom	Signature
Dumitru POTOP-BUTUCARU		Pierre-Loïc GAROCHE	
Robert DE SIMONE		Reinhard VON HANXLEDEN	
Timothy BOURKE			

Les membres du jury attestent avoir pris connaissance de l'intégralité du rapport. La Direction de la thèse atteste ne pas avoir pris part à la décision. Si le rapport comporte plusieurs pages ou s'il est rédigé sur un document distinct, il devra être paraphé sur chaque page et signé par le Président du jury. Enfin, si le rapport est rédigé dans une langue étrangère, la traduction française devra apparaître sur ce même document.

Les travaux de thèse de Monsieur Fabien Siron concernent la définition de la sémantique d'un langage temps-réel basé sur le temps logique et la mise en oeuvre de techniques de vérification formelles sur ce langage. La contribution principale est la proposition d'un cadre formel unifiant les langages synchrones et ceux à paradigme de temps logique.

Ces travaux ont été mené sur un langage industriel déjà existant. Cependant, Fabien Siron a réussi à en définir les fondations sémantiques et à aller jusqu'au développement d'une méthodologie de vérification formelle applicable à des systèmes représentatifs de l'état de l'art académique et couvrant les besoins industriels.

Les propositions théoriques de Fabien Siron ont été matérialisées dans des prototypes sérieux. Ces derniers serviront à guider l'industrialisation des fonctionalités de vérification proposées.

La présentation était claire, pédagogique et à la portée de tous. De plus, les supports de présentation soutenaient le discours tout en apportant des illustrations graphiques pertinentes.

Les questions des membres du jury furent nombreuses et Fabien Siron fut capable d'y apporter des réponses précises. Il y a montré un recul sur ses propositions et une maitrise de l'état de l'art.

Pour toutes ces raisons, le jury, uniname, lui décerne le grade de Docteur de l'Université de Côte d'Azur.

Le Président de jury précise que le directeur et le codirecteur de thèse n'ont pas participé à la délibération.

SIRON Fabien

Année universitaire 2022-2023

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

UNIVERSITÉ CÔTE D'AZUR

DOCTORAT DE L'UNIVERSITÉ CÔTE D'AZUR
AVIS DU JURY SUR L'ARCHIVAGE ET LA DIFFUSION DE LA THÈSE SOUTENUE

Titre de la thèse : Méthodologie de vérification formelle de propriétés temporelles pour les applications temps-réel critiques basées sur le concept de temps logique

Nom et prénom de l'auteur : Fabien SIRON

Membres du jury :
M. Dumitru POTOP-BUTUCARU
M. Pierre-Loïc GAROCHE
M. Robert DE SIMONE
M. Reinhard VON HANXLEDEN
M. Timothy BOURKE

Date de soutenance : 11 décembre 2023

Archivage et diffusion[1] de la thèse

Veuillez cocher obligatoirement une seule case :

A/ La version de soutenance peut être archivée et diffusée
☒ en l'état ou après corrections mineures, transmise dans les 3 mois[2]

B/ La thèse ne peut être archivée et diffusée qu'après
☐ corrections majeures demandées et validées par le jury, transmise dans les 3 mois réglementaires (la délivrance du diplôme étant conditionnée au dépôt de la thèse corrigée[3])

Nom, Prénom et Signature du président du jury : Pierre-loïc Garoche

Fait à Paris le 11/12/2023

1. La diffusion sur internet est soumise à la seule autorisation de l'auteur
2. Passé ce délai la version légale est la version déposée avant soutenance
3. Article 24 de l'arrêté du 25 mai 2016 modifié par l'arrêté du 26 août 2022

SIRON Fabien

UNIVERSITÉ CÔTE D'AZUR

Université Côte d'Azur
28 Avenue de Valrose
06108 Nice CEDEX 2

Année universitaire 2022-2023

NOTE A L'ATTENTION DU JURY

Serment doctoral d'intégrité scientifique

Le serment des docteurs relatif à l'intégrité scientifique est le suivant :

"En présence de mes pairs.
Parvenu à l'issue de mon doctorat en 'INFORMATIQUE', et ayant ainsi pratiqué, dans ma quête du savoir, l'exercice d'une recherche scientifique exigeante, en cultivant la rigueur intellectuelle, la réflexivité éthique et dans le respect des principes de l'intégrité scientifique, je m'engage, pour ce qui dépendra de moi, dans la suite de ma carrière professionnelle quel qu'en soit le secteur ou le domaine d'activité, à maintenir une conduite intègre dans mon rapport au savoir, mes méthodes et mes résultats."

"In the presence of my peers.
With the completion of my doctorate in 'INFORMATIQUE', in my quest for knowledge, I have carried out demanding research, demonstrated intellectual rigour, ethical reflection, and respect for the principles of research integrity. As I pursue my professional career, whatever my chosen field, I pledge, to the greatest of my ability, to continue to maintain integrity in my relationship to knowledge, in my methods and in my results."

- **Désignation du Président du jury :**
  « Les membres du jury désignent parmi eux un Président et, le cas échéant, un rapporteur de soutenance. Le Président doit être un professeur ou assimilé ou un enseignant de rang équivalent. ». La Direction de la thèse ne peut pas présider le jury.
- **La Direction de la thèse :**
  La Direction de la thèse assiste à la discussion et sa participation demeure précieuse pour la bonne compréhension des travaux qu'elle a encadrés. Elle peut, le cas échéant, éclairer les débats menant à la décision. Elle n'a donc pas vocation à mener les débats et, si elle assiste à la délibération, ne prend pas part à la décision finale. La Direction de la thèse est donc prise en compte dans les ratios qui peuvent être considérés au sein du collège doctoral pour les membres internes ou externes à l'établissement de rattachement. La Direction de la thèse ne signe pas le procès-verbal de délibération, mais signe le rapport de soutenance.
- **Membres invités :**
  Les membres invités peuvent poser des questions. **Ils ne peuvent en aucun cas participer aux délibérations.**
  Ils ne signent pas les documents de soutenance.
- **Visioconférence :**
  La Direction de la thèse remet la procuration du membre du jury en visioconférence au Président du jury. Le Président signe en son nom le procès-verbal et le rapport de soutenance.
- **Délibérations :**
  « Dans le cadre de ses délibérations, le jury apprécie la qualité des travaux du doctorant, leur caractère novateur, l'aptitude du doctorant à les situer dans leur contexte scientifique ainsi que ses qualités d'exposition. Le jury peut demander des corrections [...]. »

Documents de soutenance :

La Direction de la thèse ou le Président du jury a la responsabilité de remettre les documents de soutenance originaux à l'école doctorale concernée, dûment complétés et signés, au plus tard 15 jours après la soutenance :

- Le procès-verbal de soutenance signé par l'ensemble des membres du jury
- Le rapport de soutenance signé par le Président et contresigné par l'ensemble des membres du jury
- L'avis du jury sur la reproduction numérique de la thèse signé par le Président.
- La « procuration membre de jury en visioconférence » le cas échéant.

Apogée

Université Côte d Azur

Edition d'attestations de réussite

Demandeur: TOKENDEDJAM N OKENDE-DJAMNA Tshoha

UNIVERSITÉ CÔTE D'AZUR

ATTESTATION DE REUSSITE AU DIPLOME

Le Directeur des Études et de la Formation atteste que

le Doctorat

a été décerné à

Monsieur FABIEN SIRON

né le 23 janvier 1996 à CLICHY (092)

au titre de l'année universitaire 2022/2023

Discipline	:	INFORMATIQUE
Titre des travaux	:	Méthodologie de verifications formelle de propriétés temporaires pour les applications temps-réel critiques basées sur le concept de temps logique/ Methodology for the formal verification of temporal properties for real-time safety-critical applications based on logical time
Date de soutenance	:	11 décembre 2023
Etablissement soutenance	:	UNIVERSITÉ CÔTE D'AZUR
Jury	:	M. Pierre-Loic GAROCHE, Président du jury, PROFESSEUR DES UNIVERSITÉS ENAC TOULOUSE M. Reinhard VON HANXLEDEN, Rapporteur du jury, PROFESSEUR D'ÉTABLISSEMENT ÉTRANGER UNIVERSITÉ DE KIEL M. Thimothy BOURKE, Membre du jury, PROFESSEUR DES UNIVERSITÉS UNIVERSITÉ PARIS DAUPHINE - PSL M. Robert DE SIMONE, Co-Directeur, DIRECTEUR DE RECHERCHE UNIVERSITÉ CÔTE D'AZUR M. Dumitru POTOP-BUTUCARU, Directeur de thèse, CHARGÉ DE RECHERCHE INRIA PARIS
Ecole doctorale	:	SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Fait à Nice, le 19 janvier 2024

Pour le Président d'Université Côte d'Azur
et par délégation

UNIVERSITÉ CÔTE D'AZUR
27.07.2019

Le Directeur des Études et de la Formation

Pascal CREMOUX

N° étudiant : 22022251

Avis important: Il ne peut être délivré qu'un seul exemplaire de cette attestation. Aucun duplicata ne sera fourni.

LE GOURRIEREC Geoffrey
Responsable de la majeure GISTRE à l'EPITA
14-16 rue Voltaire
94270 Le Kremlin-Bicêtre
Email: geoffrey.legourrierec@epita.fr
Tél.: 07 82 54 79 60

Le 16 janvier 2024

Objet: Attestation d'enseignements réalisés à l'EPITA par Mr. SIRON Fabien

A qui de droit,

Je soussigné, LE GOURRIEREC Geoffrey, responsable de la majeure GISTRE ("Génie Informatique des Systèmes Temps Réels et Embarqués") atteste que Mr. SIRON Fabien a effectué au cours des années 2022-2023 et 2023-2024 35 heures (cours magistral et travaux pratiques) comme prévu par ses contrats d'enseignant vacataire (CFO). J'atteste également de sa participation aux jurys de stage de fin d'études de plusieurs étudiants.

Concernant l'enseignement, ces heures ont été effectuées au sein de deux modules d'enseignement en GISTRE:

- ARINC, un cours de M1 d'introduction à la programmation synchrone pour remplir des contraintes temps réel dur avec la norme ARINC-653 comme illustration;
- TEST, un cours de M2 portant sur des méthodologies, approches et outillages concrets traitant la testabilité et de la vérifiabilité d'un système logiciel.

Les heures ont été réparties comme suit:

Cours	Année	CM	TD	TP	Examen	Suivi	Total heures
ARINC (M1)	2022	2	0	2	0	0	4
ARINC (M1)	2023	2	0	2	0	0	4
TEST (M2)	2022	9	3	3	0	0	15
TEST (M2)	2023	7,5	1,5	3	0	0	12

Concernant les jurys de soutenances de stage, Mr. SIRON participa en tant qu'assesseur les 24, 25 et 26 juillet 2023 pour un total de 27 soutenances. De fait, il contribua à:

- L'évaluation des rapports de stage;
- L'évaluation de l'exercice de soutenance à proprement parler, en échangeant avec le restant du jury et avec l'entreprise d'accueil du stagiaire.

Mr. LE GOURRIÉREC
Geoffrey
Responsable GISTRE
Fait le 16/01/2024 à Versailles,

Semantics foundations of PsyC based on synchronous Logical Execution Time

Fabien Siron
Krono-Safe
Université Côte d'Azur, Inria
France
fabien.siron@krono-safe.com

Dumitru Potop-Butucaru
Robert de Simone
dumitru.potop_butucaru@inria.fr
robert.de_simone@inria.fr
Inria
France

Damien Chabrol
Amira Methni
damien.chabrol@krono-safe.com
amira.methni@krono-safe.com
Krono-Safe
France

ABSTRACT

Task models for Real-Time Scheduling (RTS) and Synchronous Reactive (SR) languages are two prominent classes of formalisms for the design and analysis of time-critical embedded systems. Task models allow providing deadlines, periods, or other such kinds of interval time boundaries that make the system description fit for schedulability analysis. Synchronous reactive languages use logical clocks to be activation condition triggers in languages providing programmability. We consider here synchronous LET (sLET) extensions that intend to re-use notions of logical clocks and logical time, for the purpose of providing schedulability boundaries. As its name indicates, sLET borrows deeply from Logical Execution Time ideas, where timing dimensions are all provided at logical design time, but they extend asynchronous events as in xGiotto with SR-inspired programmability and "first-class citizen" logical clock constructs. Our work results in a two-level semantics of the programming language PsyC. The benefits are to reuse techniques from both RTS and SR. Big-step RTS models provide inputs for task model schedulability analysis and implementation. Meanwhile, SR small-step models provide methodological tools to view any events as a time base (logical clock) and verification technologies (but they do not consider the WCET of tasks to be kept within time boundaries by the scheduling). We show the semantic equivalence of those two semantics at visible time interval boundaries.

CCS CONCEPTS

• **Computer systems organization** → **Real-time languages**; **Real-time system specification**.

KEYWORDS

Multiform Logical Time, Synchronous languages, Logical Execution Time, Real-Time Systems

ACM Reference Format:
Fabien Siron, Dumitru Potop-Butucaru, Robert de Simone, Damien Chabrol, and Amira Methni. 2023. Semantics foundations of PsyC based on synchronous Logical Execution Time. In *Cyber-Physical Systems and Internet of Things Week 2023 (CPS-IoT Week Workshops '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3576914.3587495

CPS-IoT Week Workshops '23, May 09–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0049-1/23/05…$15.00
https://doi.org/10.1145/3576914.3587495

1 INTRODUCTION

Real-Time Systems have to handle time in a predictable manner. Specified temporal requirements from a system specification should still hold in the final system. Thus, time must be considered from the very start of the design. However, exact computation durations are usually not available in early design phases as they depend on the target. In answer, various formalisms based on the Multiform Logical Time concept have been introduced to abstract [4] those target-dependent durations (expressed in physical time) with logical time constraints (expressed in logical time). Then, specific analysis, usually called schedulability analysis (or time safety analysis) can be used to fill the gap between logical and physical time, that is, they ensure that physical computations satisfy their logical time constraints. In RTS models this usually translates into the fact that real-time constraints are specified as *requirements*, while worst-case execution time (WCET) of tasks is provided as guarantees. Schedulability analysis then breaks down in satisfying the contract that WCET guarantees imply real-time constraints. Although the PsyC language defined later includes WCET modeling and scheduling resolution, it is out of the scope of the current paper, which focuses on its specification expressiveness combining influences from RTS, SR and LET.

The synchronous approach introduced a discretized abstraction of time based on logical clocks in which computations and reactions happen in discrete atomic instants, and so, logically instantaneously [5]. While multiple logical clocks may be used for specification expressiveness of sophisticated event-based timing patterns, the traditional operational semantics of synchronous languages usually imposes to expand the behaviors on a unique parent clock. Consequently, compilation of synchronous languages may suffer from the "Long Task Problem" [12]. When different tasks of different rhythms are compiled, physical execution time variability is usually limited to a common cycle rate. More recently, the *Logical Execution Time* (LET) paradigm [13] has been introduced to give a compromise between the strong expressiveness of the synchronous approach and the efficiency of traditional task scheduling. For that, LET mandates to specify the actual logical duration a task has to fulfill based on a uniform pseudo-physical time. This forms LET intervals in which communications can only happen on its bounds. Inputs can only be consulted at the start of the interval and outputs are displayed to other tasks at the end. Consequently, as communication can only be made at predefined instants, LET ensures the *temporal determinism* property.

Previous work has introduced early results on the definition of synchronous LET (sLET) as being a variation of LET based on

the Multiform Logical Time approach [19]; it allows defining LET intervals with respect to multiple logical clocks. Therefore, the duration of an interval is specified as being *up to the next n^{th} tick of a clock*. It should be noted that such logical duration is not necessarily constant across all reactions as it is specified *relatively* to a clock. In the simple case where there is only one global clock used as in the original LET paradigm, all durations become constants. Hence, sLET is a generalization of the original LET paradigm. Moreover, as the sLET allows multiple valid schedules taking into account physical time that are mutually equivalent with respect to logical clocks, the classical synchronous semantics is actually one of those in which computations happen logically instantaneously at the start of sLET intervals and outputs are delayed at the end. This article goes further than our previous work in [19], introducing an equivalence relation theorem between semantics.

To illustrate this approach, this article gives in section 3 the abstract syntax and the formal semantics of PsyC, an industrial language developed by the company *Krono-Safe*, which implements the sLET paradigm. The approach may benefit from the fact that it is an industrial-scale language, with a user community of embedded engineers. The formal semantics of PsyC is then defined using two approaches, one native semantics in section 3.2 and an other semantics defined by translation into the Esterel language in section 4, thereafter called the *synchronous semantics*. Both semantics are observationally equivalent with respect to sLET interval boundaries formalized in section 4.3. Indeed, while the synchronous semantics covers a whole PsyC application, our native semantics covers only individual agents (i.e. PsyC sequential tasks). Through a discussion in section 5, this article shows how synchronous techniques such as formal verification could then be re-used for (s)LET based languages.

2 RELATED WORK

The synchronous approach has been implemented in languages such as Esterel or Lustre [5]. It is based on the Multiform Logical Time approach and thus, totally abstracts execution time to focus on logical instants, allowing both determinism and concurrency. Nonetheless, compilation can be non-trivial given non-negligible (physical) execution times, as described in the *Long Task Problem* to which some answers have been proposed [12]. Still, today, the common workaround in the industry is to slice long tasks into sub-tasks that fit in the instant period.

The Logical Execution Time approach has been introduced with the Giotto language [13], later extended with the Timing Definition Language (TDL), which allows describing applications with a fixed set of periodic tasks and some global modes mechanism. However, those two languages do not consider logical time as being anything else than a simple abstraction of physical time. Similarly to PsyC, TimedC extends C with timing primitives, although also using an abstraction of physical time [15]. xGiotto [11] is closer to our work as it extends Giotto with events handled by a mechanism called event scoping. While our sLET paradigm can express the same kind of patterns, it treats any event or time(s) basis in the same way through logical clocks.

sLET also shares similarities with the Sparse Synchronous Model (SSM) [10]; both dedicated to applications with sparse and potentially irregular computations. However, sLET still handles time with logical time units as in the Multiform Logical Time approach while SSM handles temporal expressions based on abstracted physical time units. Nonetheless, as SSM, the sLET semantics is also inspired by discrete event model such as Lingua Franca [14].

Although related, sLET also differs from k-periodic networks or N-synchronous systems [8]. These formalisms aim at providing a certain flexibility in exact execution instants based on a global timing framework in which synchrony can be relaxed. In sLET the focus is extended to placing in time behaviors with a certain duration in terms of number of instants spent in a single behavior.

3 THE PSYC LANGUAGE

3.1 Informal Description of PsyC

The industrial language PsyC is a language developed by the French company Krono-Safe [1], which provides a set of tools for the design and the integration of safety-critical real-time applications. Such applications can then be certified at the highest level of criticality for the avionic domain (DAL-A with the DO-178C standard). PsyC stands for *Parallel SYnchronous* and has been initially presented as a model based on the timed-triggered approach [9]. As stated in the introduction, previous work has introduced an early definition of the synchronous LET paradigm as being a generalization of the LET paradigm. As the modern version of PsyC can now use multiple logical clocks, it naturally implements the synchronous LET paradigm.

Similarly to Multiform Logical Time concept, time is defined through the use of logical clocks, that is, totally-ordered sequences of ticks. The PsyC language allows describing two levels of logical clocks:

- *sources* are externally-defined logical clocks, they can be mapped to a timer (i.e. pseudo-physical time) or any events (e.g. the rotation of an engine crankshaft);
- *clocks* define a sub-sampling of sources; they are defined through an affine relation $p \times c \quad o$ with p and o being respectively the period and the offset with respect to another *clock* or *source* c.

```
1 body start {
2   y1 = f(x1);
3   advance 2 with A;
4   y2 = g(x2);
5   advance 3 with B;
6 }
```

Listing 1: Simple PsyC example

A PsyC application is composed of multiple concurrent components, called *agents*. Each agent defines an infinite sequential behavior using a syntax based on the C language. A special statement called *advance* allows synchronizing on a tick of a given PsyC clock. They specify both the deadline of preceding code, and the activation of code following it. Moreover, following the LET semantics, the *advance* specifies the instants in which communication can happen. As an example, the Listing 1 describes an infinite loop

application	::=	*decl**
decl	::=	*source*
	|	*clock*
	|	*temporal*
	|	*agent*
source	::=	source c
temporal	::=	temporal $id\ =\ v$ with c
clock	::=	clock $c_1 = n_1 \times c_2 \quad n_2$

agent	::=	agent id (starttime n with c) *body*
body	::=	body id *stmt*
stmt	::=	$id\ :=\ f(exp*)$
	|	$stmt_1$; $stmt_2$
	|	while exp do *stmt*
	|	if (exp) $stmt_2$ else $stmt_3$
	|	advance n with c
	|	next b
	|	skip

Figure 1: Abstract syntax of our PsyC subset

of two functions $f()$ and $g()$ in which the former interval takes 2 ticks of clock A while the latter takes 3 ticks of clock B.

agents can communicate with each other through dedicated deterministic communication channels. The main one, called *temporal variable*, is an implicit one-to-several real-time data flow. The task owner of the temporal variable updates this flow at a predetermined rhythm. Moreover, its value is sampled with respect to a *clock* allowing an additional sampling level between different *agents*, but we will not consider it in this article. Determinism of communication is ensured by the visibility principle, defined as follows:

- a data can only be timestamped with a date greater or equal to its corresponding deadline;
- a data can be consulted only if its timestamp is lower or equal to the current activation date;

In the Listing 1, in the first interval, $y1$ is made visible at the deadline specified by the next advance at line 3 while in the second interval, $x2$ should be visible from the activation specified by the preceding advance, also at line 3.

```
1  source source_ms;
2  clock c20ms = 20 * source_ms;
3  clock c50ms = 50 * source_ms;
4
5  agent GNC(starttime 0) {
6    consult sensors, mode; display commands;
7    body start {
8      if (mode == NOMINAL) {
9        commands = GNC(sensors);
10       advance 2 with c20ms;
11     }
12     advance 1 with c50ms;
13   }
14 }
```

Listing 2: PsyC implementation of a GNC task [6]

This mechanism, along with agent synchronization, is an implementation of the synchronous Logical Execution Time approach. As a simple example, consider the Listing 2. It implements a modified version of the conditional triggering of a Guidance, Navigation and Control System (GNC) as described in [6]. The example uses three clocks: a source clock, expected to have a period of `1 ms`, and two periodic clocks `c20ms` and `c50ms` which have a period of respectively 20 and 50 `ms`. The example shows only one agent called `GNC` which has two *consulting* inputs: `sensors` and `mode`, and one

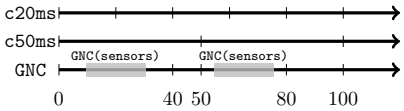


Figure 2: Possible timeline of the `GNC` task in *nominal* mode.

displaying output: `commands`. When *not in* nominal mode, the condition of line 8 is false and the agent waits for the next tick of clock `c50ms`. When *in* nominal mode, the condition of line 8 is true and the computation is constrained with a deadline of 2 ticks of clock `c20ms`. Then, afterward, the agent waits for the next tick of clock `c50ms` as shown in Figure 2.

In this article, we consider a subset of PsyC which grammar is described in Figure 1. The left column specifies the PsyC constructions defined at the application level while the right column describes the PsyC syntax at the agent level. The syntax is quite abstract with respect to the concrete one which is based on the C language. However, the objective here is to highlight the reactive part of PsyC.

3.2 Native Operational Semantics of PsyC

This section gives a native formal semantics of PsyC agents based on the Structural Operational Semantics (S.O.S.) approach introduced by Plotkin [16] and adapted later by Berry [17] to reactive languages. A more detailed version of the semantics will be made available in a research report [18]. The global approach is to successively rewrite the program such that each rewriting represents a logical instant. Transition rules (or relations) describe valid rewritings of the program. The following section describes the notations used by these rules.

3.2.1 Configuration and Transitions syntax. In this paper, a configuration (i.e. a program state) of an agent is defined by the following tuple:

$$\langle E, b, T_{output} \rangle$$

where E is the private environment of the agent, b is the identifier of the next body to be executed and T_{ouput} is the public environment of the agent. This allows to distinguish values that can be accessed by other tasks (through the temporal variable mechanism) and values that shouldn't be accessed.

Two different transitions are considered in the semantics: transitions that explicitly consume logical time and ones that are executed

in the instant. We shall call the former *temporal transitions* and the latter *non-temporal transitions*.

Non-temporal transitions are expressed using the following syntax:

$$C \vdash t \longrightarrow C' \vdash t'$$

where C (resp. C') denotes the agent configuration before (resp. after) the transition and t (resp. t') denotes the agent program before (resp. after) the transition. Temporal transitions are defined similarly:

$$C \vdash t \Longrightarrow_{n \times s} C' \vdash t'$$

where $n \in \mathbb{N}^*$ and s is a source denoting a temporal transition that has a duration of n ticks of the source s.

Moreover, to form sLET intervals, we can combine non-temporal transitions followed by a temporal one:

$$\frac{C \vdash t \longrightarrow C_1 \vdash t_1 \longrightarrow \ldots C_{n-1} \vdash t_{n-1} \Longrightarrow C_n \vdash t_n}{C \vdash t \Longrightarrow C_n \vdash t_n}$$

3.2.2 Sources and Clocks. While sources are considered as inputs in the semantics, PsyC clocks can be defined using three operators:

- *Source*(c) which gives the source from which clock c is derived;
- $\Pi(c)$ which gives the absolute period (in source ticks) of clock c with respect to its source;
- $\Phi(c)$ which gives the absolute offset (in source ticks) of clock c with respect to its source.

Additionally, to avoid keeping an unbounded date for each source, we assume that d_c gives the (current) date of c modulo its period. It is defined as $d_c = (d_s - \Phi(c))\ mod\ \Pi(c)$ with respect to its corresponding source date d_s.

3.2.3 Semantics rules of agent. The three following rules describe the basic statements of PsyC. skip does not make time progress and is rewritten to itself while the assignment and the next are defined in the same way but update their environment accordingly.

$$C \vdash \mathsf{skip} \longrightarrow C \vdash \mathsf{skip} \qquad \text{(nothing)}$$

$$E, b, T \vdash x := f(exp\ ) \longrightarrow E[x \leftarrow [\![\ f(exp\ )\ ]\!]_E], b, T \vdash \mathsf{skip} \qquad \text{(assign)}$$

$$E, b_1, T \vdash \mathsf{next}\ b_2 \longrightarrow E, b_2, T \vdash \mathsf{skip} \qquad \text{(next)}$$

The advance statement however is more complex. It performs the two following steps:

- it makes time progress with a duration computed using the corresponding clock state d_c.
- and it updates the available outputs to the ones computed during the interval.

$$\frac{N = n \times \Pi(c) - d_c \qquad T' = \mathrm{UpdateOutputs}(E) \quad s = \mathrm{Source}(c)}{E, b, T \vdash \mathsf{advance}\ n\ \mathsf{with}\ c \Longrightarrow_{N \times s} E, b, T' \vdash \mathsf{skip}} \qquad \text{(advance)}$$

Based on these basic rules, control statements can be described easily. Rules if-1 and if-2 describe the rewriting of the condition statement when respectively the condition expression is true or false. Similarly, rule while-1 and while-2 describe the rewriting of the `while` statement. Finally, rule seq describes the rewriting of the sequence statement which rewrites its first part until it is terminated. Then, it is rewritten to its second part.

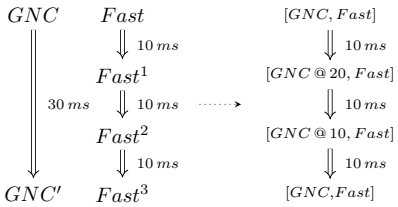


Figure 3: Agent network semantics

$$\frac{[\![\ exp\ ]\!]_E \quad 0}{E, b, T \vdash \mathsf{if}\ (exp)\ s_1\ \mathsf{else}\ s_2 \longrightarrow E, b, T \vdash s_1} \qquad \text{(if-1)}$$

$$\frac{[\![\ exp\ ]\!]_E = 0}{E, b, T \vdash \mathsf{if}\ (exp)\ s_1\ \mathsf{else}\ s_2 \longrightarrow E, b, T \vdash s_2} \qquad \text{(if-2)}$$

$$\frac{[\![\ exp\ ]\!]_E = 0}{E, b, T \vdash \mathsf{while}\ exp\ \mathsf{do}\ s \longrightarrow E, b, T \vdash \mathsf{skip}} \qquad \text{(while-1)}$$

$$\frac{[\![\ exp\ ]\!]_E \quad 0}{E, b, T \vdash \mathsf{while}\ exp\ \mathsf{do}\ s \longrightarrow E, b, T \vdash s\ ;\ \mathsf{while}\ exp\ \mathsf{do}\ s} \qquad \text{(while-2)}$$

$$\frac{C \vdash s_1 \longrightarrow^* C' \vdash \mathsf{skip}}{C \vdash s_1\ ;\ s_2 \longrightarrow C' \vdash s_2} \qquad \text{(seq)}$$

The statement rules defined above can then be used to describe the semantics of agents. For that, body can be rewritten to a loop containing a sequence of `if` statements (one for each body). The research report [18] describes the whole semantics of the PsyC language considering more advanced body constructs like early exit (e.g. equivalent to Esterel trap).

The parallel product is very similar to the synchronous one in the general case considering that the start and the end of sLET interval are synchronous instants. However, when all clocks are based on a unique source, then it can be defined as a sequence of intervals with some duration in which each agent is either in a global state (i.e. the start or the end of one of its interval) or in an intermediate state due to the overlapping with an other agent as illustrated in Figure 3.

4 SYNCHRONOUS SEMANTICS OF SLET

4.1 Description

The global idea comes from the following observation: for a given task, the sLET paradigm is equivalent to a synchronous model in which the communication model is delayed. By moving up all computations to the start of the sLET interval, the small-step semantics masks the ability to dispatch them all along. The main role of small-step semantics is therefore to allow the reuse of development environments based on SR languages. More generally, part of this work must be understood as an attempt to draw a precise semantic link between *existing* languages, emphasizing the place of synchronous LET in the making. The synchronous translation

is based on the following pattern: 1) on the activation instant, the inputs are read, the computation is done synchronously, and the output is saved in an internal state; 2) we then wait for the specified duration; and 3) finally, the outputs generated during the interval are displayed and made visible to other tasks on the last instant.

4.2 Structural Translation of PsyC to Esterel

To illustrate the approach above we propose in this article a structural translation of PsyC to the synchronous language Esterel. We chose Esterel because its syntax is very close to PsyC. It's mainly control-flow and imperative. In particular, the advance statement is very similar to the Esterel `await` statement. While a data-flow synchronous language such as Lustre has more available and up-to-date tools today for analysis, the translation of PsyC to Lustre is far more complicated as the control-flow needs to be flattened. However, the circuit semantics of Esterel describes a data-flow representation of its semantics, which could be represented in Lustre.

The global idea of the translation is to represent PsyC clock and source ticks with Esterel signals and PsyC temporal variables with Esterel valued signals as well as Esterel variables to handle private agent copies. Based on that, agent can be translated, almost as is, with PsyC advance statement translated to the Esterel `await` statement. The translation rules are as follows:
PsyC skip is just translated by:

```
nothing
```

PsyC assignation of temporal x, $x := exp$ is translated by:

```
private_temporal_x := T(exp)
```

PsyC assignation of variable x, $x := exp$ is translated by:

```
var_x := T(exp)
```

PsyC `next` statement, next b, is translated by:

```
next_body := b
```

PsyC statement's advance, of the form advance n with c and assuming `var1`, `var2` ... `varN` are all the temporal variables in output of the agent, is translated by:

```
await n c;
emit temporal_var1(private_temporal_var1);
emit temporal_var2(private_temporal_var2);
...
emit temporal_varN(private_temporal_varN);
```

Control structures are translated into their equivalent form in Esterel without any difficulties, so they are not given here. The global translation scheme of an agent then directly results from the translation patterns defined above. Its module interface is composed of temporal variables (inputs and outputs) which are translated to Esterel valued signals and PsyC clocks which are translated to Esterel pure signals. As an illustration, the Listing 3 shows how the PsyC implementation of the task *GNC* described in Listing 2 is translated using the rules above.

```
1  var private_commands : t_cmd in
2    loop
3      if ?mode = NOMINAL then
4        private_commands := GNC(?sensors);
5        await 2 c20ms;
6        emit commands(private_commands);
7      end if;
8      await 1 c50ms;
9    end loop
10 end var
```

Listing 3: Esterel translation of task GNC

4.3 Equivalence relation with the native semantics

Based on the introduced PsyC semantics and its Esterel translation, this section gives an observational equivalence between them for individual agents. First of all, let us define the Esterel operational semantics coming from [17] as following:

Definition 4.1 (Esterel semantics). Given a constructive Esterel program P and a set of $data$, the semantics of its macro-step is given by the following relation:

$$P, data \xrightarrow[E_i]{E_o}\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow P', data'$$

where E_i and E_o are respectively the input and output signal set active on the current reaction.

The equivalence theorem yields naturally from both semantics (PsyC and Esterel) and some data equivalence relation. Considering a sLET interval, the theorem states that both the PsyC and the Esterel representation have an equivalent behavior on the boundaries of the interval. The PsyC semantics yields only one transition while the Esterel semantics yields a sequence of unitary transitions (i.e. with respect to some source). If both data representations are equivalent at the start of the interval, then, they are also equivalent at the end. An extended proof will be available in the report [18].

Theorem 4.2. *For all PsyC agent p_{ag} and the Esterel translation T and for any agent transition of the form:*

$$C \vdash p_{ag} \Longrightarrow_{n\times s} C' \vdash p'_{ag}$$

Assuming, $C \approx data$ and $s \in E$, we have an equivalent sequence of Esterel transitions:

$$T(p_{ag}), data \xrightarrow[E]{}\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow P^1, data^1 \ldots \xrightarrow[E]{}\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow P^n, data^n$$

with $T(p'_{ag}) = P^n$ and $C' \approx data^n$

Proof. By structural induction on native rules structure □

5 DISCUSSION

As mentioned earlier, the dual semantic presentation (native and synchronous) was meant to target two distinct further design paths: real-time scheduling models lead to efficient multicore *compilation* based on schedulability analysis; synchronous reactive extensions yield verification techniques mostly based on model-checking of synchronous models. Note that, currently, each path is somehow weak for covering the other goal: SR expansion do not preserve WCET features and computations become instantaneous; RTS models do not exactly fit with existing verification formalisms and need

further encoding. We now comment briefly on experiments in these two directions.

5.1 Compilation

Currently, the efficient PsyC compilation chain restricts to *mono-source* programs, corresponding to a single global clock (as in Lustre or LET), but where affine clock down sampling is still allowed in multi-rate specifications. The compiler may then flatten all interval durations on the single source, and then checks real-time schedulability along with WCET inputs. A last issue remains when the actual interval value is data-dependent, as in the following case:

```
if (...)
  advance 1 with c10ms;
else
  advance 3 with c10ms;
```

The current compiler treats this case by looking for a uniform solution that satisfies the shortest delay (i.e. 10 ms), relaxing this constraint afterward (i.e. 30 ms when going to the "else" branch). However, we view this problem has a general issue for future work, especially in the case of multi-source systems.

5.2 Verification

Since PsyC systems cannot deadlock or support clock preemptions, verification efforts consist mainly in establishing the correctness of logical timed properties such as latencies. These are expressed as synchronous observers, in our case using the Clock Constraint Specification Language CCSL [3]. In the "simple efficient compilation case" as described above, verification models such as simple Timed Automata or even Integer Linear Programming solvers could be considered. However, in the general case the mix of boolean logic and integer duration constraints will require mixed-techniques, such as SAT/SMT or BDD. As ongoing work, we focused so far primarily on NuSMV [7] and Prover PSL model-checkers [2]. PSL is an industrial model-checker, developed by Prover, heavily used to demonstrate safety of railway systems.

Partial results are listed in Figure 4 conducted on two simple but representative PsyC case studies (both composed of 5 agents) from a benchmark suite coming from [20]: an Anti-Lock Braking System (ABS) and a Temperature Control System (TCS). The results for both tools are satisfying, even in the more complicated setting of ABS1-3 in which a slow mode involves long durations. More advanced results are expected in future work.

Requirement	NuSMV (s)	Prover PSL (s)
TCS1 *(Causality)*	4.223	3.005
TCS2 *(Periodicity)*	1.897	0.037
TCS3 *(Latency)*	0.149	0.022
ABS1 *(Causality)*	13.365	13.674
ABS2 *(Periodicity)*	0.687	11.777
ABS3 *(Latency)*	2.344	61.14

Figure 4: Verification results in seconds of a subset of requirements of two use-cases coming from [20].

6 CONCLUSION

We showed how semantic background from Synchronous Reactive Languages and Real-Time Scheduling task models could respectively allow providing a small-step and big-step meaning to PsyC, or alternately how the PsyC language could be seen as an exploitation of these notions in a commercial framework. We showed the two semantics to coincide on agent synchronization points (i.e. interval boundaries). These ideas are strongly indebted to the Logical Execution Time design philosophy, emphasizing the potential use of sporadic events as logical clocks syntactically involved in reactive language constructs. We hope this common ground will help better understand efficient multicore compilation as well as efficient timed verification and analysis in the future.

REFERENCES

[1] 2023. *Krono-Safe.* https://www.krono-safe.com/
[2] 2023. *Prover Technology.* https://www.prover.com/
[3] Charles André. 2010. *Verification of clock constraints: CCSL Observers in Esterel.* Research Report. INRIA.
[4] Charles André, Frédéric Mallet, and Marie-Agnès Peraldi-Frati. 2007. A multiform time approach to real-time system modeling; Application to an automotive system. In *2007 International Symposium on Industrial Embedded Systems.* 234–241.
[5] Albert Benveniste, Paul Caspi, Stephen Edwards, Nicolas Halbwachs, and Robert de Simone. 2003. The Synchronous Languages 12 Years Later. *Proc. IEEE* 91 (2003).
[6] Thomas Carle, Dumitru Potop-Butucaru, Yves Sorel, and David Lesens. 2015. From Dataflow Specification to Multiprocessor Partitioned Time-triggered Real-time Implementation *. *Leibniz Transactions on Embedded Systems* (2015).
[7] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. 2000. NuSMV: a new symbolic model checker. *International journal on software tools for technology transfer* 2, 4 (2000), 410–425.
[8] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. 2006. N-synchronous Kahn networks: a relaxed model of synchrony for real-time systems. *ACM SIGPLAN Notices* (2006).
[9] Vincent David, Jean Delcoigne, Evelyne Leret, Alain Ourghanlian, Philippe Hilsenkopf, and Philippe Paris. 1998. Safety Properties Ensured by the OASIS Model for Safety Critical Real-Time Systems. In *SAFECOMP.*
[10] Stephen A Edwards and John Hui. 2020. The sparse synchronous model. In *2020 Forum for Specification and Design Languages (FDL).* IEEE, 1–8.
[11] Arkadeb Ghosal, Thomas Henzinger, Christoph Kirsch, and Marco Sanvido. 2004. Event-Driven Programming with Logical Execution Times. In *International Workshop on Hybrid Systems: Computation and Control*, Vol. 2993. 357–371.
[12] Alain Girault and Xavier Nicollin. 2003. Clock-Driven Automatic Distribution of Lustre Programs. In *Embedded Software.* Springer, 206–222.
[13] Christoph Kirsch and Ana Sokolova. 2012. The Logical Execution Time Paradigm. In *Advances in Real-Time Systems.*
[14] Marten Lohstroh, Christian Menard, Soroush Bateni, and Edward A Lee. 2021. Toward a Lingua Franca for deterministic concurrent systems. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 4 (2021), 1–27.
[15] Saranya Natarajan and David Broman. 2018. Timed C: An Extension to the C Programming Language for Real-Time Systems. In *Real-Time and Embedded Technology and Applications Symposium.* Los Alamitos, CA, USA, 227–239.
[16] Gordon Plotkin. 2004. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.* (2004).
[17] Dumitru Potop-Butucaru, Stephen A Edwards, and Gérard Berry. 2007. Constructive Operational Semantics. *Compiling Esterel* (2007), 79–102.
[18] Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni. 2022. *Formal Semantics of the PsyC language.* Research Report. INRIA Sophia Antipolis - Méditerranée (France). to appear.
[19] Fabien Siron, Dumitru Potop-Butucaru, Robert De Simone, Damien Chabrol, and Amira Methni. 2022. The synchronous Logical Execution Time paradigm. In *ERTS 2022 - Embedded Real Time Systems.* Toulouse, France.
[20] Jagadish Suryadevara, Cristina Seceleanu, Frédéric Mallet, and Paul Pettersson. 2013. Verifying MARTE/CCSL mode behaviors using UPPAAL. In *International Conference on Software Engineering and Formal Methods.* Springer, 1–15.

The synchronous Logical Execution Time paradigm

Fabien Siron*†, Dumitru Potop-Butucaru†, Robert de Simone†, Damien Chabrol* and Amira Methni*
*Krono-Safe, Massy, France
†Université Côte d'Azur, Inria, France
*firstname.lastname@krono-safe.com
†firstname.lastname@inria.fr

***Abstract*—Real-Time industrial systems are not so much of those that have to perform tasks incredibly fast, but in a time-predictable manner; they rather focus on meeting previously specified timing requirements in a provable way. Consequently, time must be taken into account from the very start of the design. However, exact timing constants may not be available yet in early design stages as they may depend on the target. In answer, formalisms based on the Multiform Logical Time have been introduced to abstract real-time durations. The Synchronous-Reactive (SR) approach introduced a discretized abstraction of time on which computations happen logically instantaneously. Contrary to SR, Logical Execution Time (LET) mandates to specify the actual logical duration a task has to fulfill. This allows a more efficient compilation, at the price of a lower expressiveness. Classical LET (i.e. as introduced in *Giotto/TDL*) sticks to uniform pseudo-physical time, i.e. based on one logical clock mapped to the real-time. In this paper, we introduce a new paradigm called *synchronous Logical Execution Time* (sLET) that builds upon both SR and LET paradigms. It keeps the idea of *logical durations* coming from the LET paradigm, while having logical instants based on logical clocks. This extends the expressivity of LET, as time is totally abstracted as sequences of events. The various schedulings provide physically timed versions that, while having distinct non-functional properties (in terms of performance mostly), remain mutually functionally equivalent (in the logical time realm). A particular instance, where computations are executed "in a single instant", and then time is advanced (as in classical event-driven simulation), can lead to a direct translation into synchronous formalisms (in our case *Esterel*). We started inquiring how this could open new ways of verification and analysis on PsyC programs.**

I. Introduction

The design of embedded control software calls for stringent real-time constraints due to their permanent interaction with the physical environment. Such real-time systems are usually safety-critical because of the context in which they are used (avionic, automotive ...). Therefore, their designs should be verified with the highest possible level of confidence. Various formalisms based on the concept of *Multiform Logical Time* have been introduced to abstract real-time durations which are usually not known at the design phase, as they might be target-dependent. Then, specific analysis, called schedulability analysis (or time safety analysis) ensure that physical computations satisfy their logical time constraints.

Among those formalisms, the *Logical Execution Time* (LET) paradigm [1] brings a compromise between the strong expressiveness of the synchronous-reactive approach (SR) [2] and the efficiency of traditional task scheduling. Contrary to the SR model, computation takes time and is bounded by a fixed logical duration usually known before the computation happens. However, in SR model, because computations are considered instantaneous, time can be refined using multiple logical clocks. LET is usually based on a unique clock representing the progress of time. Consider the following temporal interval where the bounds are described by `sync`:

```
sync 1 ms; f(); sync 1 s;
```

Depending on the semantics of the language, the interval might last one second or up to the next second tick. The former semantics, usually used by LET based languages, would mean that both ms and s refer to the same clock (i.e. $1\,s$ being just a short for $1000\,ms$) while the latter semantics, corresponding to synchronous languages, mean that both ms and s are logical clocks related with an affine relation.

In industrial projects, while classical synchronous languages such as Lustre or Scade fit perfectly the need to describe the functional behavior of the system, Synchronous LET, which is implemented by the industrial language *PsyC* [3], actually focus on a different level of the system life-cycle, namely, the software integration level. This approach is quite classical (see [4], [5] and[6]) and divides the functional design of the system from its non-functional design, that is, how different functional components (either designed with synchronous languages or manually) are integrated. This multilayer approach is depicted in figure 1. This is, however, still a challenge today to verify that properties specified at system-level (i.e. in the specification) still holds at integration level.

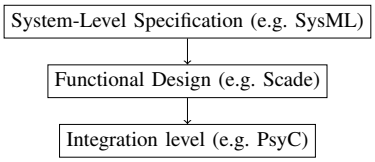


Fig. 1: Typical Software Life-Cycle of industrial real-time systems

In this paper, we introduce a new paradigm called *synchronous Logical Execution Time* (sLET) that builds upon both

the SR model and LET. It keeps the idea of *logical durations* coming from the LET paradigm while having logical instants based on logical clocks. This extends the expressivity of LET as time is totally abstracted as sequences of events. Thus, it inherits 1) the simple compilation and fine-grained schedulability analysis coming from LET and 2) the synchronous semantics that is well-suited for formal verification.

After an overview of existing Multiform Logical Time models, we introduce sLET as an extension of LET in section II. To illustrate sLET, we then give an overview of the formal semantics of a subset of the industrial language *PsyC* [3] in section III, developed by the company Krono-Safe, and then, we give translation rules to the synchronous language Esterel in section IV. While *PsyC* is compiled using techniques similar to the ones of LET, formal verification techniques coming from synchronous languages such as *Esterel* could then be re-used. Such approach is illustrated in section V where *PsyC* and Scade interact in a classical use-case showing a flight control system.

II. Multiform Logical Time Models

Synchronous languages introduce the notion of Multiform Logical Time, through logical clocks, so they could better be called Polychronous time. A logical clock counts the successive ticks of any relevant event. Still, their operational semantics expands the behaviors in terms of a grandmother clock, the discrete reaction step measuring the instant. On the other hand, initial LET formalisms rely on such a totally ordered discrete time, and introduce quantitative durations measured in it; although they may use pseudo-physical unit names (milliseconds, nanoseconds,...) for it, the compliance with actual physical implementation is only a wish (or a requirement), that will have to be checked later when the latter becomes available. We now extend on these notions, and the abstraction they allow for fast and human-legible logical design, cleanly split from the later physical implementation (and without back and further adjustements hopefully, a main purpose of these formalisms). The purpose is to position the sLET formalism as combining multiform logical time and the ability of user-defined logical clocks, together with quantitative durations in which a certain amount of computation can be spread in any instants of a given interval, providing it does not overflow it.

A. The Synchronous-Reactive Model (SR)

The Synchronous-Reactive model, implemented by synchronous languages [2], totally abstracts execution time to focus on logical instants, allowing both determinism and concurrency. For that, computations react simultaneously and instantaneously with the tick of a global common logical clock as shown in figure 2a, which can be further refined to give multiple logical clocks. The synchronous hypothesis, then, ensures that if every computation is bounded by the next reaction, then physical time can be safely ignored as shown in figure 2b. Combining synchrony and concurrency allows a very expressive formal model while keeping it very simple [2]. Hence, SR model is well adapted for formal verification.

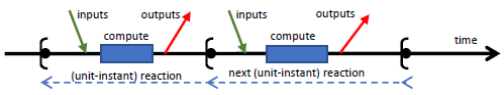
SR model has been introduced in languages such as *Esterel* [7] which is mainly imperative and *Lustre* [8] which is declarative. The compilation of synchronous languages is, however, quite complex. First, instantaneous communication makes the compilation for parallel platforms (i.e. multicore, distributed) quite hard and can produce causality issues. Secondly, execution time is usually limited to the reaction time of the system. Thus, applications with non negligeable execution time (i.e. also called the long period problem) can be rejected during the schedulability analysis. Nonetheless, languages based on the synchronous model such as *Prelude* [6], can adress this problem, but are usually limited to specific patterns such as a multi-periodic synchronous model.

B. The Logical Execution Time Model (LET)

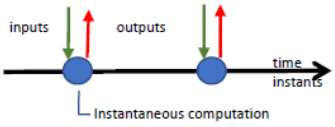
The Logical Execution Time paradigm abstracts physical time through a sequence of instants, similarly to the synchronous approach. However, contrary to the latter, computation takes time and is not considered instantaneous. For that, each computation must fit in a logical interval, called LET interval [1]. Furthermore, communications are only made on the boundaries of LET intervals. Inputs are read at their beginning and outputs are made visible to other tasks at their termination (see Figure 2c). As communication is only done on predefined instants, computations behave like they always take the same time, which is the LET interval duration (see Figure 2d). As such, the determinism property is ensured.

The LET paradigm has been initially introduced in the *Giotto* [1] language in which each task correspond to a periodic LET interval. The language can also express global modes that allows to change from a given set of task frequency to an other. *Timing Definition Language* (TDL) [9] extends *Giotto* via a decomposition in concurrent modules. Each of them defines a set of task and their frequency, similarly to *Giotto*, but a module can also change its local mode independently of other modules. *Giotto* has also been extended with events in the *xGiotto* language [10] through the mechanism of event scoping. In all these approaches, the bounds of LET intervals are either defined using chronometric pseudo-physical durations or events (i.e. in *xGiotto*). This paper proposes a more abstract and homogeneous model of time, that is, the use of multiple logical clocks.

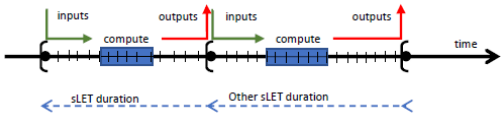
In a way, LET extends logical time with the concept of logical durations. This allows a precise schedulability analysis due to an increased execution time variability. In turn, this also makes compilation to parallel platforms easy and causality issues of the SR model are avoided. However, as LET forbids



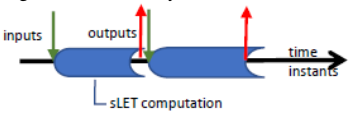
(a) Physical view of the Synchronous-Reactive model



(b) Logical view of the Synchronous-Reactive model



(c) Physical view of the (synchronous) Logical Execution Time model



(d) Logical view of the (synchronous) Logical Execution Time model

Fig. 2: Multiform Logical Time models: physical vs logical views

instantaneous communications, its theoretical expressivity is reduced compared to the SR model.

C. sLET as a synchronous extension of LET

Synchronous Logical Execution Time (sLET) extends the classical LET paradigm with the concept of logical clocks. As such, interval bounds can be triggered by clock ticks while in classical LET, interval duration is usually defined by a constant fixed duration. However, as in LET, communication can only happen on interval bounds, which ensure temporal determinism. Being closer to the SR model, sLET is actually fully compatible with the synchronous hypothesis, that is, computations can happen instantaneously on the activation date of the interval as long as outputs are delayed to the end of the interval. This model is actually a generalization of the *Psy* model introduced in the OASIS framework since the 90's [3].

A logical clock, in the sense of Lamport [11], abstracts time through a series of events called *clock ticks*. The sLET paradigm adds to LET a finite set of logical clocks $\mathbb{C}$ on which LET interval boundaries can be based. We will call *clock event* an event of the form $n \times c$ where $n \in \mathbb{N}^*$ and $c \in \mathbb{C}$. This event basically means that we have to wait n ticks of the clock c. An interval termination is then defined with a *clock event* which may also define the activation of a next interval. As an example, if an interval termination is defined with $n_t \times c_t$, then after its activation, it should wait for n_t ticks of the clock c_t. Note that if all intervals use the same clock, then clock events actually express fixed constant durations with respect to that clock which is similar to the classical LET model.

sLET keeps with the LET idea of imposing at an early phase of logico-functional design some fixed interval durations at which boundaries the I/O external behaviors will be supposed to occur. Then there is flexibility on when exactly computations are scheduled, as long as they remain inside these bounds. Next, we will introduce the PsyC languages foundations, bringing syntax to the sLET view. In particular, PsyC allows conditional durations, in alternative "if-then-else" behaviors. Note however that classical LET, as defined in *Giotto*, can actually be expressed easily with sLET, that is, using only one logical clock mapped to real-time. The converse is not true. Indeed, even considering that all logical clocks are derived using affine relation from a unique global clock, sLET interval might have different duration depending on the current global state.

III. The PsyC language: abstract syntax and formal semantics

A. Language description

In this section, we give a brief description of *PsyC*, a language developed by the company Krono-Safe, as an illustration of the sLET paradigm. This language is implemented in the ASTERIOS product which provides a set of tools to design safety-critical real-time software. Such applications can then be certified at the highest level of criticity for the avionic domain (DAL-A, DO-178C). ASTERIOS and *PsyC* are inherited from the OASIS and PharOS projects coming from the CEA. Initially, *PsyC* (for *Parallel SYnchronous*), in OASIS, was presented as a timed-triggered approach for safety-critical real-time software under a model called *Psy* [3]. This model is actually very close to LET, and synchronous LET generalize both of them. The modern version of *PsyC* is actually better characterized with synchronous LET due to the existence of multiple logical clocks.

A *PsyC* application is composed of a fixed set of tasks called *agents* that are formed of sLET intervals. The content of *agents* is composed of C code that is extended with special instructions like `advance` n `with` c which specifies the boundaries of sLET interval. Informally, its semantics is to advance the logical time of n ticks of a clock c. Such clocks have to be defined in the application as an affine relation with

respect to another clock. They are actually two kinds of logical clocks in *PsyC*, *sources*, which are defined externally, and *clocks*, which basically refine *sources*. In practice, most of the applications define only one *source* called *realtime* which is mapped on real time. The Figure 3 shows an example of an agent using multiple clocks and a conditional computation in the *PsyC* language. A possible timeline is shown in figure 4. Such pattern with multiple clocks allowed by sLET couldn't be expressed, as is, in the classical LET model.

```
source realtime;
clock c2 = 2 * realtime;
clock c3 = 3 * realtime;

agent Ag(uses realtime, starttime 1 with c2)
{
  body start /* infinite loop */
  {
    /* computation A */
    advance 1 with c3;
    if (/* condition */) {
      /* another computation B */
      advance 1 with c2;
    }
  }
}
```

Fig. 3: Example of a PsyC agent

Inter-agent communication is performed through a dedicated channel called *Temporal Variable*. Multiple agents can read a *temporal variable* but only one can write into it. According to sLET, inputs and outputs in a sLET interval can only be made on sLET interval bounds. Hence, data emitted by an agent is made visible (i.e. the temporal variable is updated) on the instant described by the next `advance` instruction. Futhermore, *Temporal Variables* also have another layer of sampling (defined with a clock) and values are read with the expression $\$[n]var$ which denotes the n^{th} last sampled value.

The Figure 5 describes the abstract syntax of a subset of *PsyC* which is necessary to introduce the formal semantics described in the next section. Only specific *PsyC* constructions are described and the syntax of C expressions is omitted for simplicity. Also, we add a specific statement skip that does nothing (i.e. in the C syntax, this could represented as a semi-

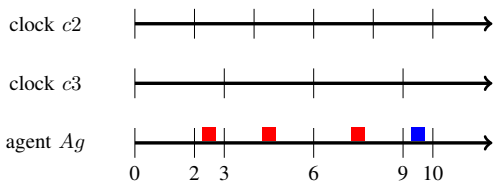


Fig. 4: Possible timeline of example in fig 3. A Red frame represent computation A while a blue frame represent computation B.

application	::=	$decl^*$
decl	::=	source c
	|	clock $c_1 = n_1 * c_2 + n_2$
	|	temporal id = $value$ with c
	|	*agent*
agent	::=	agent id $body^*$
body	::=	body id $stmt$
stmt	::=	id := exp
	|	skip
	|	advance n with c
	|	$stmt_1$; $stmt_2$
	|	if (exp) $stmt_2$ else $stmt_3$
	|	...

Where $n, n_1 \in \mathbb{N}^*$, $n_2 \in \mathbb{N}$, v is the initial value of the temporal variable and c_i are clock identifiers.

Fig. 5: Abstract syntax of our *PsyC* subset

colon). This will be helpful in the semantics to express that a statement doesn't have any work left to do. Futhermore, both the `starttime` and the source parameter of the agent are omitted because the former could be defined as an `advance` and the latter could be deduced from the clocks. However, we assume that an agent could only use clocks that are derived from a unique source. Nonetheless, different agents might use different sources.

B. Formal Semantics

In this section, we introduce the formal semantics of a subset of the *PsyC* agents thanks to term rewriting. Basically, a term can be rewritten sucessively, with respect to a given set of formal rules, to form a symbolic execution that serves as an oracle. The rules are given using the Structural Operational Semantics (S.O.S.) approach introduced by Plotkin [12] and later adapted to synchronous languages [7]. The idea behind S.O.S. is to give the rewriting relation of a term with respect to the rewriting of its parts. This is usually defined through a set of inference rules. The behavioral semantics of Esterel extends S.O.S. through the use of an integer encoding the type of the transition, i.e. whether the transition takes time or is instantaneous [7] (i.e. fits entirely inside an instant reaction).

As described above, the semantics of *PsyC* is described in this paper using two relations, one that makes time progress, and one that is not explicitely timed, and should be considered as conceptually guaranteed to fit in the time interval closed by the next time-progress behavior. This vision can be considered as borrowed from discrete-event simulation models. While the semantics allows many later schedules for the internal behaviors before next time advance (a property exploited by the *PsyC* compiler to optimize their time allocations according to other criteria), one may also consider the (valid) interpretation where all computations are made in the initial instant, than

will allow the translation to synchronous languages of the next section.

Consider that the configuration of an agent (i.e. its state) can be described as a pair $\langle E, T\rangle$ where E is the assignements of its local variables and T the assignments of its displayed temporal variables (i.e. its outputs). Based on this, we define an instantaneous transition as following:

$$\langle E, T\rangle \vdash s \longrightarrow < E', T' > \vdash s'$$

This transition denotes that a statement s can be rewritten in s' instantaneously while the configuration $< E, T >$ is updated to $< E', T' >$ with respect to s. Based on this definition, we can define the semantics of the first basic statement, the assignation:

$$\langle E, T\rangle \vdash x \ := \ exp \longrightarrow \langle E', T\rangle \vdash \mathsf{skip} \qquad \text{(assign)}$$

where $E' = Update(E, v, exp)$.

The $assign$ rule evaluates its expression and updates its local variables accordingly using a function that we call $Update()$. This statement is logically instantaneous because it is executed at the beginning of a sLET interval to read the correct input values. Also, it is reduced to skip as there is no work left to do. The $advance$ rule is a little bit more complicated as this statement makes time progress. Recall however that clocks used in an agent by advance statements are all derivated from a unique source. Thus, we first instantaneously rewrite the $advance$ statement with respect to its parameters (i.e. number of ticks and clock) to a counter of source ticks that we define with the pseudo statement *Remains*. This pseudo statement takes an absolute duration in parameter (with respect to the agent source) that is computed with the function $Duration$ with respect to the $advance$ parameters, and the current source date.

$$\langle E, T\rangle \vdash \mathsf{advance}\ n\ \mathsf{with}\ c \longrightarrow \langle E, T\rangle \vdash \textit{Remains}(N) \qquad \text{(advance)}$$

where $N = Duration(n, c, date)$

The *Remains* pseudo statement defined above can then be used to make time progress. For that, we define a new transition that takes one time unit of the agent source (symbolized by the double arrow shape):

$$\langle E, T\rangle \vdash s \Longrightarrow < E', T' > \vdash s'$$

Remains is then defined with two different rules. The first one, *Remains-1*, decreases the counter when its greater than 1. When the counter is equal to 1, *Remains* is rewritten to skip as there is no work left to do and temporal variables are updated. As other agents can only read the latter (and not the agent local environment), this actually means that outputs can only be made visible at the end of a sLET interval, which is actually the semantics of the sLET paradigm.

$$\langle E, T\rangle \vdash \textit{Remains}(N) \Longrightarrow \langle E, T\rangle \vdash \textit{Remains}(N-1) \qquad \text{(Remains-1)}$$

if $N > 1$

$$\langle E, T\rangle \vdash \textit{Remains}(1) \Longrightarrow \langle E, T'\rangle \vdash \mathsf{skip} \qquad \text{(Remains-2)}$$

where $T' = UpdateOutputs(T, E)$.

The rules defined above can be composed quite easily. First, when there are executed in sequence, multiple instantaneous transitions can be represented with one big instantaneous transition and when they are followed by a non instantaneous transition, they can be represented with a big non instantaneous transition. In other words, the semantics of an agent can be represented with only big non instantaneous temporal transitions, which is interesting to show the expected temporal behavior of the agent. Second, the rules defined above only show simple statements, but actually, this can be extended very easily to control flow statement (e.g. condition statement) given that the type of transition is propagated correctly.

To illustrate a little bit the semantics above, we show a very basic example of a sLET interval with an output variable called tv. Let's consider the following PsyC statements: $tv \ := \ 2$; advance 2 with $c2$ Let's assume that $c2$ is a strictly periodic clock with period 2 and offset 0, and the current date of the agent is odd. Then, with respect to the semantics rules, we have the following execution:

$$\begin{aligned}
&\langle tv =?, tv =?\rangle \vdash tv \ := \ 2 \ ; \ \mathsf{advance}\ 2\ \mathsf{with}\ c2 \\
&\longrightarrow \langle tv = 2, tv =?\rangle \vdash \mathsf{advance}\ 2\ \mathsf{with}\ c2 \\
&\longrightarrow \langle tv = 2, tv =?\rangle \vdash \textit{Remains}(3) \\
&\Longrightarrow \langle tv = 2, tv =?\rangle \vdash \textit{Remains}(2) \\
&\Longrightarrow \langle tv = 2, tv =?\rangle \vdash \textit{Remains}(1) \\
&\Longrightarrow \langle tv = 2, tv = 2\rangle \vdash \mathsf{skip}
\end{aligned}$$

This execution shows that the communication is correctly updated at the end of the interval and its duration is actually 3 as it depends on the current date (e.g. if the date was even, then the interval should have lasted 4 ticks). Also, one can notice that such pattern is not possible in classical LET languages (e.g. *Giotto*, *TDL*) as they usually can only represent duration on a single clock.

Construction	PsyC syntax	Esterel Translation
Clock	$clock\ c = n_1 * c_p + n_2$	$await\ immediate\ n_2;$ $loop$ $\quad emit\ c$ $each\ n_1\ c_p$
Temporal Variable	$temporal\ tvar\ =\ init\ with\ c$	$signal\ tvar = init\ in$ $\quad run\ Sampler[tvar/In, tvar_0/Out, c/Clk]\ \|\|$ $\quad ...$ $endsignal$
Local Variable	$int\ var\ = init;$	$var\ private\_var\ :=\ init\ in\ ...$
Assignation/expression	$tvar = exp;$	$private\_tvar\ :=\ exp;$
Temporal variable reading	$\$[0]tvar$	$tvar_0$
Condition	$if\ (exp)\ s1\ else\ s2$	$if\ (exp)\ s1\ else\ s2$
Advance	$advance\ n\ with\ c$	$await\ n\ c; emit\ tvar(private\_tvar)$
Body (cyclic case)	$body\ p$	$loop\ p\ end$
Agent composition	$agent\ ag1\ ...agent\ ag2\ ...$	$run\ ag1[...]\ \|\|\ run\ ag2[...]\ \|\|\ ...$

Fig. 6: Some Esterel translation patterns (var is a private variable and $tvar$ is a (shared) temporal variable)

IV. Consequences

A. Synchronous semantics of sLET via Esterel translation

To show the direct relation between sLET and the SR model, we can encode the semantics of *PsyC* into a synchronous language like *Esterel*. We choose *Esterel* because its syntax is quite close to *PsyC*, that is, both languages are control-flow and imperative. Basically, the approach is to focus on the logical behavior of synchronous LET and to totally abstract actual executions, i.e. to consider them (logically) instantaneous. Thus, of course, this kind of simulation is not representative of the actual operational behavior where each computation takes time. Nonetheless, sLET (similarly to classical LET) allows multiple valid schedules that are mutually equivalent, that is, that form an equivalence class with respect to logical clock ticks. The synchronous semantics is then actually a particular instance where in a sLET interval:

1) all computations are executed during the first instant, thus inputs are read synchronously to the start of the interval;
2) time is advanced to the end of the interval;
3) outputs are produced synchronously to the end of the interval.

This abstraction should be sufficiant to express properties at the model level as actual computations should not impact temporal properties of a sLET design. While this approach focus on synchronous LET, as the latter is a generalization of the classical LET model, this approach could be adapted easily to other LET languages.

Esterel is an imperative synchronous language control-flow based. In particular, it allows imperative concurrency, that is, handling multiple instruction pointers at a same instant and preemption where a computation can be aborted when a given signal is present. The only communication means available in *Esterel* is through instantaneous signals which can have a status, *present* or *absent*, and a value. A signal which only has a status is said to be *pure*, otherwise, it is said to be *valued*. The instructions are basically divided into two categories similarly to *PsyC*:

- the instantaneous instructions, like $emit\ s$ which emits a given signal s;
- and temporal instructions, like $pause$ or $await\ s$ which respectively wait for the next instant and for the next instant where signal s is present.

For a full definition of the Esterel language, we invite the reader to consult [7].

In a *PsyC* application, different elements are actually composed in a synchronous parallel fashion. Assuming that *PsyC* clock ticks are represented as *Esterel* pure signals (i.e. either the clock ticks or not), we have to represent clock generation, temporal variable sampling and agent behavior. As explained at the beginning of section III, a *PsyC source* clock is actually an external logical clock, and is thus represented as an *Esterel* pure signal input. Clock generation basically refines a clock tick, i.e. a pure signal, based on an affine relation. A temporal variable is represented as a persistent valued signal (i.e. which is always present) and is sampled on the signal corresponding to its clock. And finally, an agent is almost translated as is. The control-flow is translated in its equivalent form in *Esterel* and basic *body* patterns can be translated either to sequence or to infinite loop depending on the next body to be executed. We do not cover advanced *body* patterns in this paper for simplicity but this could be adapted with some more advanced *Esterel* control-flow patterns. All variables of an agent, either local to the agent or temporal variables, are translated to *Esterel*

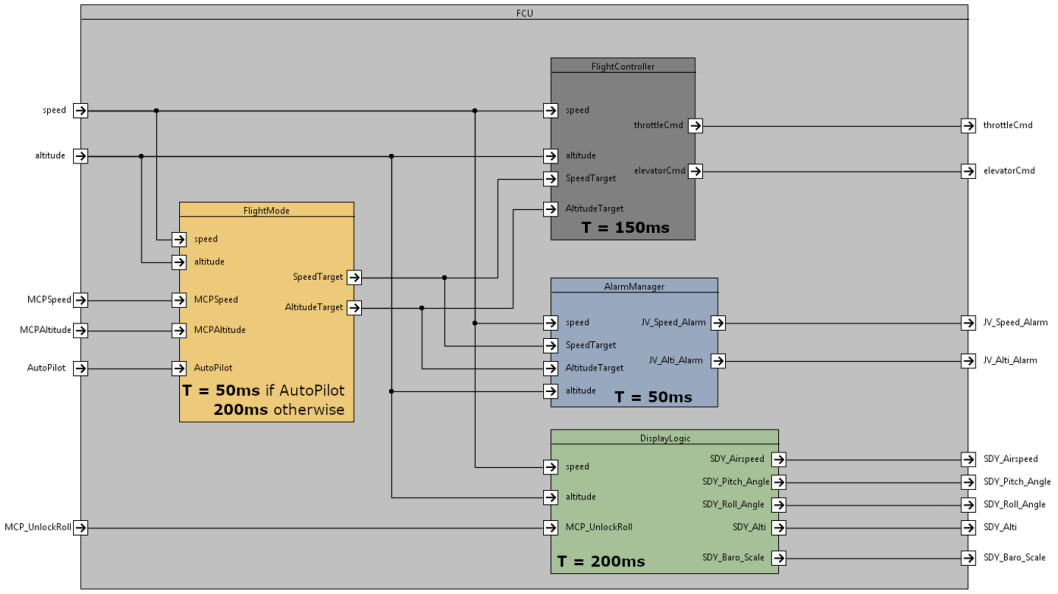


Fig. 7: The Flight Control System use-case

variables. The *advance* statement is translated to the *await* statement followed immediately by an update of the temporal variables, that is, the valued signal of each temporal variable in output of the agent is updated with its local variable value. This is actually the main difference from the synchronous model, communication cannot be updated instantaneously. All the translation patterns described above are sketched in the table 6.

B. Synchronous Observers and Formal Verification

The main interesting consequence of the *Esterel* translation is to apply techniques and tools coming from the synchronous community. Among them, formal verification has been successfully applied on the synchronous approach due to two main things: 1) the synchronous composition semantics, the state space is usually quite reduced comparing to asynchronous approaches and 2) symbolic model-checking allows to scale verification techniques even more, using BDD or SMT techniques. Futhermore, in synchronous languages, properties that are usually modeled using temporal logics, can also be modeled directly in the language using the so-called synchronous observer approach [13]. Observers only observe the state of the application and defines which state is acceptable or not. Classicaly, a signal *Error* is emitted if an error is detected and verification is reduced to a reachability analysis on this signal.

In *PsyC*, the typical kind of property that we might want to check applies on the different temporal variables rhythms. Typically, due to the sampled buffered semantics of temporal variable, if the producer and the consumer are not at the same rhythm, produced values can be lost or consulted values might be duplicated. While the second scenario is usually not a problem, the first one might be one. Of course, this kind of property is quite simple, but this opens the possibility to more evolved properties like freshness constraints of temporal variables or end-to-end latencies.

V. Use-Case

A. Description

As highlighted in the introduction, languages based on LET, such as *PsyC*, focus on the architecture description at integration level of real-time systems. In other words, they allow to express how multiple functional components connect and what is their real-time behavior. In particular, it is possible to express multi-rate behavior. Typically, in industrial applications, a synchronous language such as Scade, is used to design the functional components of the system and a language like *PsyC* can be used to specify the real-time software integration of these components.

To reflect this approach, the following use-case is based on an example found in the Scade Suite software. Consider a basic

Flight Control Software in which you expect the following behavior:

- given an altitude and a speed command, a flight controller implements control laws for altitude and speed with respect to altitude and speed sensors;
- depending on the flight mode, i.e. AutoPilot or Manual, the commands given to the flight controller is either the input commands or the commands of the previous instant;
- regulation alarm is raised if the altitude sensor (respectively the speed sensor) is too far from the altitude command (respectively the speed command).

The system architecture is shown in the figure 7. Basically, each component correspond to a function detailed above. Each functional component can either be implemented manually or with a *Scade* sheet. To illustrate a little bit more this approach we show in figure 8 the *Scade* sheet corresponding to the mode handling. The mode handling is modeled with a finite state machine with only two states, the *AutoPilot* state and the *Manual* state corresponding to the *AutoPilot* mode and *Manual* mode described at the beginning of the section. The control laws are based on PID controllers but we will not dive into the details as it is beyond the scope of this paper.

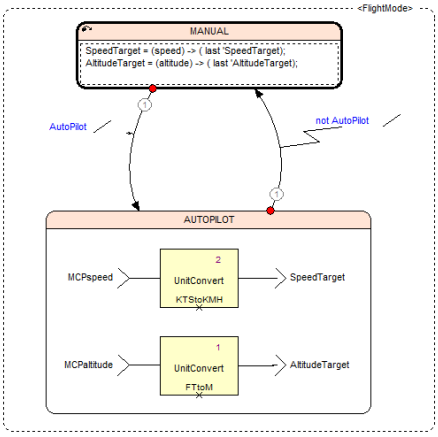


Fig. 8: Mode Controller of FCS

For simulation purpose, the example also contains a plane simulation that takes throttle and elevation commands (from the controller), simulates the actuator response as well as the aerodynamics behavior and yield updated speed and altitude sensors values (i.e. to the controller). Running such a simulation in *Scade* typically ignores real-time constraints and every components run at the same rate. While this is totally justified for simulation, it's is not necessarily the case for compilation. Indeed, in practice, different components might have different real-time behaviors, depending on available resource, as well as different mapping constraints (typically for multicore target). Thus, in the next section we will show how *PsyC* can be used to integrate seamlessly multi-rate functional components.

B. Architecture Description in PsyC

We choose a very simple mapping where each components described above are directly mapped to an individual agent. Futhermore, each communication signal is mapped to an individual temporal variable. Of course, while only one agent can write a temporal variable, multiple agents can read it. Thus, a temporal variable is not duplicated for each reader component. The temporal behavior of a component in an agent is the following:

- The reset function of the component generated from Scade is executed once with a given phase constraint (not specified here) and;
- The cycle function of the component generated from Scade is executed indefinitely *after* the reset function, on a given rate.

As an illustration, the code in 9 describes the temporal behavior of the controller component. The inputs and outputs of the cycle function are transfered with the help of global variables which is the usual calling convention with code generated from Scade.

```
agent FlightController(starttime 1 with clk_starttime)
{
  body start {
    next cycle;
    FlightController_reset_FlightControl();

    advance 1 with clk_10000_0 ;
  }
  body cycle /* infinite loop */ {
    altSensor = $[0]temporal_altitude;
    speedSensor = $[0]temporal_speed;
    speedSTarget= $[0]temporal_speedTarget ;
    altTarget= $[0]temporal_altitudeTarget ;

    FlightController_FlightControl();

    temporal_throttleCmd = throttleCmd;
    temporal_elevatorCmd = elevatorCmd;

    advance 1 with clk_150000_0 ;
  }
}
```

Fig. 9: Agent corresponding to the *FlightController* component

Chosing the right clocks is primary in the integration flow. First, the clocks should be sufficiently fast to ensure the stability of the different functional behaviors (typically control laws). Second, the clocks should be not too fast to ensure that the platform has enough resource to execute the system.

The expected clock periods are described in the figure 7. We consider that *FlightController* is a heavier task and hence, has a period three times bigger (e.g. $150ms$) than the other tasks that run with a period of $50ms$.

As explained in section III, *PsyC* also allows more advanced control flow patterns. As an example, the agent *FlightMode* may need to relax the deadline when the *AutoPilot* mode is disabled as specified in 7. This is illustrated in the listing 10. This kind of pattern doesn't have any impact on the functional behavior (i.e. commands are unchanged in manual mode), except during mode transition, and this allows to relax the cpu load.

```
agent FlightMode(starttime 1 with clk_starttime)
{
  body start {
    next cycle;
    FlightMode_reset_FlighControl();

    advance 1 with clk_10000_0;
  }
  body cycle /* infinite loop */ {
    /* From sensors */
    altitude = $[0]temporal_altitude;
    speed = $[0]temporal_speed;
    /* From user commands */
    AutoPilot = $[0]temporal_AutoPilot;
    MCPspeed = $[0]temporal_MCPspeed;
    MCPaltitude = $[0]temporal_MCPaltitude;

    FlightMode_FlightControl();

    temporal_speedTarget = SpeedTarget ;
    temporal_altitudeTarget = AltitudeTarget;

    if (AutoPilot) {
      advance 1 with clk_50000_0 ;
    } else {
      advance 1 with clk_200000_0 ;
    }
  }
}
```

Fig. 10: Agent corresponding to the *FlightMode* component with conditional deadline

C. Esterel Translation

The *PsyC* architecture given above allows efficiant compilation. However, as illustrated, the right clocks has to be used to respect the temporal behavior. For that, one can use the synchronous semantics of *PsyC*, that is, its translation to Esterel to analyze the application's behaviors. Of course, theorically, such analyze could be made in the Scade application simulating the multi-rate behavior. However, we think that such approach should be made at the integration level. Futhermore, the whole application might not be available in Scade, i.e. either coming from different teams or written manually.

The listing 11 shows the translation of the *Mode* agent. We remove the function calls for simplicity but not the communication mechanism. In the main loop, *altitude* is read instantaneously then, some computation is made and the *AltitudeTarget* is set to a private variable. Finally, after the `advance`, that is, the `await` here, this private variable is made visible through the `emit` statement. Special communication events are also added to show when communication is read and when communication is produced in this module. This will be needed later for verification.

```
module FlightMode:
[
  await clk_starttime;
  /* FlightMode_reset_FlightControl(); */
  await 1 clk_10000_0;

  loop
    emit FlightMode_Read;
    altitude := ?temporal_altitude_0;
    speed := ?temporal_speed_0;
    AutoPilot := temporal_AutoPilot_0;
    MCPspeed := temporal_MCPspeed_0;
    MCPaltitude := temporal_MCPaltitude_0;

    /* FlightMode_FlightControl(); */

    private_altitudeTarget := AltitudeTarget;
    private_speedTarget := SpeedTarget;

    if autoPilot then
      await 1 clk_50000_0;
    else
      await 1 clk_200000_0;
    emit temporal_altitudeTarget(
      private_altitudeTarget);
    emit temporal_speedTarget(
      private_speedTarget);
    emit FlightMode_Write;
  end loop
]
```

Fig. 11: Translation of *FlightMode* agent in Esterel

D. Synchronous Observers

The typical properties we might want to check with synchronous observers on such system are clock tick event ordering and clock tick event synchronizations. Typically, in *PsyC*, if the ticks of two tasks are alternating, and assuming temporal variable is at the speed of the producer, then, no value is lost. In a clock algebra such as CCSL, the predicate $c_1 \sim c_2$ states that the ticks of the logical clocks c_1 and c_2 alternate. Such constraints can be easily implemented in Esterel observers as shown in [14]. Thus, verification of such properties is then reduced to a reachability analysis on a signal representing an error state that we will call *Error* in this section.

In our example, let's consider that we want to ensure that all commands values are taken into account once by the *Alarm* agent. This functional chain is thus composed of the *FlightMode* agent, the *SpeedTarget* and the *AltitudeTarget* temporal variables, and the *Alarm* agent. Considering that the corresponding tick events have the same name, we have the following constraints to check to ensure that no values are lost:

$$\begin{aligned} \textit{FlightMode\_Write} &\sim \textit{SpeedTarget} \wedge \\ \textit{FlightMode\_Write} &\sim \textit{AltitudeTarget} \wedge \\ \textit{SpeedTarget} &\sim \textit{Alarm\_Read} \wedge \\ \textit{AltitudeTarget} &\sim \textit{Alarm\_Read} \end{aligned}$$

Based on [14], an *Esterel* observer for *Alternate* can be derivated. To fit our purpose, alternation strictness (whether two synchronous tick events could be synchronous or not) should be adapted to our communication mode. That is, the transition from event A to B in $A \sim B$ can be instantaneous but not the transition from B to A. The verification of the constraints described above can be written in instanciating the `Alternate` observer for each of the constraint with the help of parallel composition:

```
run Alternate[signal FlightMode_Write / A,
              SpeedTarget / B,
              Error/Error]; ||
run Alternate[signal SpeedTarget / A,
              Alarm_Read / B,
              Error/Error]; ||
run Alternate[signal FlightMode_Write / A,
              AltitudeTarget / B,
              Error/Error]; ||
run Alternate[signal AltitudeTarget / A,
              Alarm_Read / B,
              Error/Error];
```

In our scenario, this property is verified only when the *AutoPilot* mode is set as the *Alarm* agent might read multiple times the same values of the *FlightMode* agent when *AutoPilot* is disabled. The observer could be adapted easily to enforce alternation only when the mode is set. Also, note that this property does not enforce that all clocks should be synchronous (i.e. at the same rate). In fact, the agent *Alarm* might have some offset with *FlightMode* still without losing any value.

VI. Conclusion

This article presents a new paradigm called sLET that builds upon both the synchronous and the LET paradigm. Such unification allows to re-use formal analysis techniques coming from synchronous languages while keeping the simple compilation and efficient schedulability analysis of LET. sLET (as the classical LET paradigm) allows multiple valid schedulings that are mutually equivalent (in the logical time realm), as long as all of the computations fits in their sLET intervals (the *PsyC* compiler actually using one of these solutions also satisfying actual physical computation durations). The synchronous semantics is then actually a particular instance where computations are executed "in a single instant" and outputs are delayed. Consequently, this shows that one can use analysis based on the synchronous approach to reason on languages based on LET (e.g. *Giotto*, *xGiotto*, *TDL*) at the logical level; in particular, it unveils the possibility to re-use synchronous formal verification techniques for these languages. The sLET paradigm is implemented by the *PsyC* language which is part of the ASTERIOS framework, produced by KRONO-SAFE.

This work opens twofold perspectives: firstly, express more advanced properties, such as end-to-end latencies or data freshness, in a simpler way; secondly, synchronous verification techniques should be adapted more thoroughly to be efficient. In particular, (s)LET intervals are composed of instants that only make time progress and thus, can be optimized for verification. Finally, while our main focus is formal verification, other well known techniques can be adapted to languages based on (s)LET, such as model-in-the loop simulation, automatic test generation or test coverage.

References

[1] C. Kirsch and A. Sokolova, "The logical execution time paradigm," in *Advances in Real-Time Systems*, 2012.

[2] A. Benveniste and al, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, pp. 64 – 83, 2003.

[3] V. David and al, "Safety properties ensured by the oasis model for safety critical real-time systems," in *SAFECOMP*, 1998.

[4] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, "From simulink to scade/lustre to tta: a layered approach for distributed embedded applications," *ACM Sigplan Notices*, vol. 38, no. 7, pp. 153–162, 2003.

[5] S. Bliudze, X. Fornari, and M. Jan, "From model-based to real-time execution of safety-critical applications: Coupling scade with oasis," in *Embedded Real Time Software and Systems (ERTS2012)*, 2012.

[6] J. Forget and al, "A Multi-Periodic Synchronous Data-Flow Language," in *11th IEEE High Assurance Systems Engineering Symposium*, 2008.

[7] G. Berry, *The Constructive Semantics of Pure Esterel*, 1996.

[8] D. Pilaud, N. Halbwachs, and J. Plaice, "Lustre: A declarative language for programming synchronous systems," in *14th Annual ACM Symposium on Principles of Programming Languages*, vol. 178, 1987, p. 188.

[9] W. Pree and J. Templ, "Modeling with the timing definition language (tdl)," in *Automotive Software Workshop*. Springer, 2006, pp. 133–144.

[10] A. Ghosal, T. Henzinger, C. Kirsch, and M. Sanvido, "Event-driven programming with logical execution times," vol. 2993, 2004.

[11] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[12] G. Plotkin, "A structural approach to operational semantics," *J. Log. Algebraic Methods Program.*, 2004.

[13] N. Halbwachs and P. Raymond, "Validation of synchronous reactive systems: From formal verification to automatic testing," in *5th Asian Computing Science Conference on Advances in Computing Science*, 1999.

[14] C. André, "Verification of clock constraints: CCSL Observers in Esterel," INRIA, Research Report, 2010. [Online]. Available: https://hal.inria.fr/inria-00458847