Langage Chips : Modélisation et contrôle de systèmes distribués à base de composants

Anna Gallone
Université Marie et Louis Pasteur, CNRS UMR 6174, Institut FEMTO-ST, F-25000 Besançon, France[*]

Résumé

Ce document présente Chips, un langage de programmation dédié à la modélisation de systèmes à base de composants. Son objectif est de permettre la modélisation d'architectures logicielles auxquels une forme de contrôle en temps réel serait appliquée. Pour cela, Chips intègre des constructions syntaxiques facilitant l'expression des formalismes mathématiques issus de la théorie du contrôle. son niveau d'abstraction permet également l'introduction des concepts de programmation agrégée telle la généralisation de l'opération de *broadcast* ou d'accumulation pour être appliqué à des systèmes logiciels distribués. En compilant Chips vers d'autres langages comme BIP, il sera possible de vérifier par preuve ou *model-checking* des propriétés de résilience des systèmes conçus sur des automates équivalents aux programmes initiaux.

Mots-clefs : Chips [†]*, Systèmes à Composants, Théorie du Contrôle, Modélisation, BIP* [‡]*, Programmation Agrégée*

I. Introduction

L'omniprésence des appareils connectés dans notre environnement permet d'envisager des applications qui tireraient parti de toutes les capacités de ces appareils. Que ce soit leurs capteurs (gyroscope, thermomètre, caméra...), leurs actuateurs (haut-parleur, LEDs, écrans...) ou même leur puissance de calcul associée, chacune de ces ressource peut-être mise en commun pour la réalisation de tâches. Cependant, la coordination de tous ces processus logiciels est complexe et leurs interractions fait emerger des comportements globaux qu'il est difficile de prévoir. Le terme de système complexe sera utilisé ensuite pour décrire ces ensembles de processus communiquants, chaque processus pouvant lui-même être réalisé par des sous-systèmes (potentiellement complexes eux aussi). Les appareils composant de tels systèmes sont en perpétuel mouvement, certains y entrent, d'autres en sortent. Cela donne lieu à tout une variété de configurations. De plus, les appareils sont sujets à des modifications de leur environnement (température, vent, bande passante, etc). Si un logiciel compte tirer avantage de toutes les capacités de ce système complexe, il doit pouvoir s'adapter à la fois aux changements de configurations et à l'environnement [1]. Grâce à la théorie du contrôle, il n'est pas requis de maitriser toute cette complexité pour atteindre cet objectif.

La théorie du contrôle est une discipline de l'ingénierie qui vise à asservir en temps réel un système à son environnement : le système mesure en continu un ensemble de signaux relatifs à son environnement pour réagir aux événements tout en poursuivant un objectif donné. Cela permet par exemple de réguler la vitesse d'une voiture ou d'optimiser la distribution en eau dans un réseau citadin. Cependant, même si ces concepts sont familiers à la communauté scientifique, ils demeurent difficiles à appliquer dans le domaine de l'informatique [2]. Parmi les obstacles à cette application, on peut citer la difficulté d'établir des modèles ou équations pour les systèmes complexes étudiés ici. En effet, l'espace des configurations que peuvent prendre

∗. Ces travaux sont supportés par la subvention ANR-23-CE25-0004 (ADAPT).
†. Control of Hierarchical Interconnected Programable Systems
‡. Behaviour Interaction Priority

temporairement un réseau de processus grandit de façon exponentielle avec le nombre de sous-systèmes sous contrôle. Le design d'un seul contrôleur pour toutes ces situations est inapproprié, d'où l'intérêt de développer des outils de synthèse de contrôleurs [3], et de chercher à les appliquer en temps réel [4] pour améliorer leur qualité de service.

L'application de la théorie du contôle profiterait à de nombreux domaines de l'informatique, comme en témoignent la variété des recherches intégrant des faccultés d'auto-adaptation à leurs systèmes : Services Web, IoT, Réseaux, etc [1]. Dans le cas des systèmes à composants, d'autres problématiques émergent telles que celle de synchronisation ou encore d'allocation optimale des ressources [5]. Parallèlement, le domaine de l'*Internet of Things*(IoT) a vu naître l'*Aggregate Programming*, un paradigme pour piloter des systèmes où une multitude de processus interragissent en continu [6][7]. Le langage Chips se propose comme une application de la théorie du contrôle à des systèmes à composants qui intègre des notions d'*Aggregate Programming* afin de réduire la complexité apparente des objets qu'il permet de modéliser.

La suite de ce document présente une description détaillée des challenges que le langage Chips vise à relver en section II, puis décrit en section III les moyens (qui seront) mis en place pour efficacement atteindre ces objectifs.

II. Problématique

La théorie du contrôle est un formalisme efficace pour asservir un système simple à son environnement. Dans un système complexe, il est difficile d'appliquer directement la théorie du contrôle à l'ensemble des processus contrôlés car ceux-ci peuvent avoir des effets les uns sur les autres. Modéliser formellement de tels systèmes permet d'anticiper certains de leurs dysfonctionnements dès l'étape de conception via des méthodes et outils de *model checking*. Mais si un modèle peut s'avérer efficace pour concevoir un système robuste, réaliser ce modèle demeure une tâche fastidieuse (pour la même raison qu'un système complexe est sujet à de nombreux dysfonctionnements potentiels). Le langage Chips se positionne vis-à-vis de la conception de systèmes complexes comme un langage de description de modèles. Avec un typage fort permettant d'assurer le formalisme de la théorie du contrôle, le but de Chips est d'automatiser la génération de modèles sur lesquels effectuer des simulations numériques et vérifications de propriétés. Après des transformations assurant la conservation de ces propriétés vers d'autres langages, ces modèles Chips deviendraient les premiers artefact d'une chaîne de production d'applications distribuées correctes par construction.

III. Approche Proposée

L'objectif du langage Chips est multiple, et chacune des sous-sections suivantes s'intéresse à un aspect différent de son développement :

- Formaliser la description de modèles pour des applications distribuées (sous-section III.A)
- Générer des modèles équivalents dans un autre langage, en l'occurence, sous la forme d'automates BIP (sous-section III.B)
- Vérifier la résilience de ces applications via *model-checking*, preuve, ou à défaut, les valider par simulations (sous-section III.C)

A. Formalisme du langage

Pour permettre d'adapter efficacement les formalismes de la théorie du contrôle, Chips adopte une syntaxe proche d'autres langages synchrones comme Lustre [8] ou Heptagon/BZR [9] (voir Listing 1). Chaque composant logiciel correspond à un bloc fonctionnel avec des entrées et des sorties qu'il convient de connecter pour réaliser un modèle. Pour permettre plus de souplesse dans la définition des composants, Chips se distingue des autres langages mentionnés en ce fait qu'il n'est que partiellement synchrone : La logique interne qui convertit les signaux d'entrée en signaux de sortie peut-être définie par un algorithme séquentiel et

l'usage de variables internes, un formalisme plus proche de langages de programmation générique comme le C. Pour définir un système, Chips introduit 3 types de fonctions : le type *pure*, pour factoriser des expressions sous des symboles plus expressifs, le type *virtual* pour signifier qu'il s'agit d'une suite d'opérations requérant un espace de stockage pour être implémentée, et le type *physical* pour modéliser les composants qui font intéragir les flux de données binaires avec l'environnement par le biais de capteurs ou d'actuateurs. Ce dernier type est à mettre en lien avec des specifications du support physique utilisé pour permettre de déterminer la valeur du mot réservé *dt*, le pas de temps simulé, lors de la compilation du modèle.

Listing 1 Implémentation partielle d'un modèle de ventilateur en Chips

```
pure errorf(float expected, float received)
       -> (expected - received)

virtual pid_controller(float expected,
                       float received)
init {
    float derivative = 0; float integral = 0;
    float lasterror = 0; float error = 0;
    float out = pid(error, integral, derivative);
} then {
    error = errorf(expected, received);
    derivative = (lasterror-error)/dt;
    integral = integral+error*dt;
    lasterror = error;
    out = pid(error, integral, derivative);
} -> (out)

physical fan(float power_required)
init {
    float power = 0;
    float wind_measured = 0;
} then {
    power = max(power_required, this.power_received);
    wind_measured = function_modeling_physics(power);
} -> (wind_measured)

SYSTEM {
    pid_controller virtual_pid;
    fan the_fan;
    link virtual_pid to the_fan;
    // target value 5 chosen arbitrarily
    virtual_pid.in(5, the_fan.out);
    the_fan.in(virtual_pid.out);
}
```

Le langage sera ensuite à faire évoluer avec divers éléments. Par des structures de données plus avancées et des structures de contrôle comme des boucles, on compte en améliorer l'expréssivité et permettre de modéliser des systèmes de plus grande échelle. Ces ajouts seraient ensuite à mettre en lien avec des primitives de langage plus avancées qui introduiraient des opérations issues de l'aggregate programming. On serait alors en capacité de modéliser des systèmes s'adaptant en temps réel à des changements de configuration.

B. Génération d'automates équivalents

Afin que les modèles décrits ne servent pas simplement d'outils de descriptions, il convient de les transformer sous une forme permettant des expériementations. À cette fin, le choix a été fait de compiler les modèles Chips en langage BIP [10]. BIP est un framework de modélisation dont la compilation vers C++ est correcte par construction. Il permet de décrire des systèmes à composants en représentant les processus sous la forme d'automates auxquels sont associées des variables. Pour faire interagir ces processus et faire évoluer leurs variables, BIP définit la notion de connecteurs, des composants qui définissent quelles transitions ont lieu simultanément entre des automates distincts. Par la description de composants complexes (automates connexes regroupés sous un même identifiant), BIP rend possible la conception de systèmes logiciels hierarchiques, donc parfaitement appropriés aux systèmes complexes auxquels s'intéresse Chips. An accord avec les principes de l'ingénierie des modèles, un méta-modèle de Chips est en cours de développement. Ce méta-modèle sera convertible vers le méta-modèle de BIP grâce au langage ATLAS [11]. On déterminera alors un ensemble de règles de transformations pour traduire n'importe quelle description Chips en une implémentation à base d'automates d'état finis en BIP.

C. Simulations et preuve de propriétés

Une fois des automates équivalents aux programmes initiaux générés, deux pistes sont à explorer pour valider les modèles développés :

- Le langage BIP est doté d'un moteur de simulation. Il est donc possible d'explorer l'espace des executions possible des systèmes pour les valider, soit par explorations exhaustives de l'ensemble des états possibles des systèmes, soit par une approche statistique et des méthodes de *model checking* [12].
- Des propriétés de résilience peuvent être également prouvées sur les automates directement. On peut notamment employer les preuves relatives aux automates quantitatifs sur les systèmes décrits par BIP pour vérifier la convergence de réseaux de processus vers un état désiré [13].

À plus long terme, BIP donne l'occasion de générer automatiquement des implémentation réelles des programmes modélisés pour tout types d'appareils en conservant leur aspect distribué [14].

Références

[1] Alfonso, I., Garcés, K., Castro, H., and Cabot, J., "Self-adaptive architectures in IoT systems : a systematic literature review," *Journal of Internet Services and Applications*, Vol. 12, No. 1, 2021, pp. 1–28. https://doi.org/10.1186/s13174-021-00145-8, URL https://link.springer.com/article/10.1186/s13174-021-00145-8, number : 1 Publisher : SpringerOpen.

[2] et al., A. F., "Software Engineering Meets Control Theory," *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, edited by P. Inverardi and B. R. Schmerl, IEEE Computer Society, 2015, pp. 71–82. https://doi.org/10.1109/SEAMS.2015.12, URL https://doi.org/10.1109/SEAMS.2015.12.

[3] Arioka, Y., Yamauchi, T., and Tei, K., "Pre-controller Synthesis for Runtime Controller Synthesis," *2023 IEEE 13th Int. Conf. on Control System, Computing and Engineering (ICCSCE)*, 2023, pp. 161–166. https://doi.org/10.1109/ICCSCE58721.2023.10237143, URL https://ieeexplore.ieee.org/abstract/document/10237143/authors.

[4] Shi, H., Dong, W., Li, R., and Liu, W., "Controller Resynthesis for Multirobot System When Changes Happen," *Computer*, Vol. 53, No. 12, 2020, pp. 69–79. https://doi.org/10.1109/MC.2020.3017343, URL https://ieeexplore.ieee.org/document/9269907.

[5] Ben Halima, R., Kallel, S., Gaaloul, W., Maamar, Z., and Jmaiel, M., "Toward a correct and optimal time-aware cloud resource allocation to business processes," *Future Generation Computer Systems*, Vol. 112, 2020, pp. 751–766. https://doi.org/10.1016/j.future.2020.06.018, URL https://www.sciencedirect.com/science/article/pii/S0167739X19333679.

[6] Beal, J., Pianini, D., and Viroli, M., "Aggregate Programming for the Internet of Things," *Computer*, Vol. 48, No. 9, 2015, pp. 22–30. https://doi.org/10.1109/MC.2015.261, URL https://doi.org/10.1109/MC.2015.261.

[7] Casadei, R., Viroli, M., Aguzzi, G., and Pianini, D., "ScaFi : A Scala DSL and Toolkit for Aggregate Programming," *SoftwareX*, Vol. 20, 2022, p. 101248. https://doi.org/10.1016/J.SOFTX.2022.101248, URL https://doi.org/10.1016/j.softx.2022.101248.

[8] Halbwachs, N., "A synchronous language atwork : the story of lustre," *Proceedings. Second ACM and IEEE Int. Conf. on Formal Methods and Models for Co-Design, 2005. MEMOCODE '05.*, IEEE, Verona, Italy, 2005, pp. 3–12. https://doi.org/10.1109/MEMCOD.2005.1487884, URL http://ieeexplore.ieee.org/document/1487884/.

[9] Delaval, G., Marchand, H., and Rutten, E., "Contracts for modular discrete controller synthesis," *SIGPLAN Not.*, Vol. 45, No. 4, 2010, p. 57–66. https://doi.org/10.1145/1755951.1755898, URL https://doi.org/10.1145/1755951.1755898.

[10] Basu, A., Bensalem, S., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.-H., and Sifakis, J., "Rigorous Component-Based System Design Using the BIP Framework," *IEEE Softw.*, Vol. 28, No. 3, 2011, pp. 41–48. https://doi.org/10.1109/MS.2011.27, URL https://doi.org/10.1109/MS.2011.27.

[11] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I., "ATL : A model transformation tool," *Science of Computer Programming*, Vol. 72, No. 1, 2008, pp. 31–39. https://doi.org/10.1016/j.scico.2007.08.002, URL https://www.sciencedirect.com/science/article/pii/S0167642308000439.

[12] Legay, A., Delahaye, B., and Bensalem, S., "Statistical Model Checking : An Overview," 2010. https://doi.org/10.1007/978-3-642-16612-9_11, URL https://inria.hal.science/inria-00591593.

[13] Boker, U., Henzinger, T. A., Mazzocchi, N., and Saraç, N. E., "Safety and Liveness of Quantitative Properties and Automata," , Feb. 2025. https://doi.org/10.48550/arXiv.2307.06016, URL http://arxiv.org/abs/2307.06016, arXiv :2307.06016 [cs].

[14] Jaber, M., "Centralized and Distributed Implementations of Correct-by-construction Component-based Systems by using Source-to-source Transformations in BIP," phdthesis, Université Joseph-Fourier - Grenoble I, Oct. 2010. URL https://theses.hal.science/tel-00531082.