# Gestion de traces d'exécution pour le systèmes embarqués : contenu et stockage

Vania Marangozova-Martin, Generoso Pagano

# Gestion de traces d'exécution pour le systèmes embarqués : contenu et stockage

Vania Marangozova-Martin[*], Generoso Pagano[†]

[*]Université Joseph Fourier, LIG (équipe Nanosim)
[†]INRIA Grenoble, équipe-projet MESCAL

Ce rapport porte sur les systèmes de traces et catégorise leurs motivations et les fonctionnalités fournies. Il a pour objectif d'expliciter le lien entre objectifs de traçage et les types (contenu, format et stockage) de traces d'exécution manipulées. Il identifie les besoins en termes d'exploitation de traces dans le domaine des systèmes embarqués et présente notre proposition de solution dans le cadre du projet SoC-TRACE.

# Table des matières

# 1 Introduction

Le développement spectaculaire des systèmes embarqués présente de nouveaux défis de conception matérielle et logicielle. L'apparition accélérée de produits variés demande, en effet, de nouveaux outils qui facilitent le processus de production et de mise au point. Dans ce contexte, le projet SoC-TRACE [**?**] se focalise sur le problème de gestion et d'analyse *à posteriori* de traces d'exécution.

Tracer un système et exploiter ses traces d'exécution est un moyen classique de mise au point [**?**]. Tracer un système consiste à enregistrer des *historiques d'exécution* reflétant les *événements* qui se sont produits dans le système pendant son exécution. Les traces capturées (acquises) varient selon le type des événements considérés (logiciels, matériels), le niveau de détail de traçage, l'organisation des informations (structures et formats) et leur stockage.

Exploiter les traces d'exécution consiste à analyser les informations enregistrées afin de répondre à des questions sur le fonctionnement du système (Comment marche-t-il ?), l'évaluation des performances (Le système est-il performant ?), la validation (Respecte-t-il les spécifications ?), le débogage (Où est l'erreur ?) et l'optimisation (Comment pourrait-il marcher mieux ?).

Dans ce rapport nous nous intéressons à l'état de l'art des outils de traçage, identifions les besoins dans le domaine des systèmes embarqués et introduisons notre proposition de contenu, stockage et manipulation de traces dans le cadre du projet SoC-TRACE. Nous commençons par présenter de manière générale les aspects d'acquisition et d'analyse de traces (chapter 2). Nous discutons ensuite des systèmes de traces proposés dans différents domaines (chapter 3) et explicitons leurs supports en termes de contenu et de stockage de traces (chapter 4). Notre proposition de gestion de traces dans le cadre du projet SoC-TRACE est présenté dans la chapter 5. Nous concluons par les premiers résultats d'utilisation de notre solution et les perspectives de travail.

# 2 Enjeux des systèmes de traces [1]

Deux questions principales se posent quand il s'agit de travailler avec des traces d'exécution. La première concerne la phase d'*acquisition* pendant laquelle sont interceptées et enregistrées les informations composant la trace. La deuxième concerne la phase d'*analyse* pendant laquelle la trace est soumise à différents types de traitements afin d'en extraire de la connaissance sur le fonctionnement du système. Dans certains cas, typiquement dans le domaine du calcul haute performance [REF], l'analyse à effectuer sur les traces conditionne la manière dont elles sont produites. Néanmoins, le constat et la motivation du projet SoC-TRACE est qu'aujourd'hui la plupart des systèmes de traces, surtout dans le domaine embarqué, se focalisent sur l'aspect d'acquisition. L'aspect d'analyse reste encore peu développé et reste centré sur la visualisation scientifique où les aptitudes du développeur sont indispensables.

Dans la suite, nous présentons notre vision générale de la gestion de ces deux aspects.

## 2.1 Acquisition de traces

Les deux défis à traiter lors de l'acquisition d'une trace d'exécution sont la complétude et les performances.

- **Complétude** Pour fournir une trace qui reflète fidèlement l'exécution d'un système, le traçage doit relever un maximum d'événements, fournir suffisamment de détails et fonctionner pendant suffisamment de temps.
- **Performances** Le traçage doit induire une perturbation (intrusion) minimale au système. Il doit supporter un débit d'enregistrement élevé et pouvoir produire des traces de taille importante.

Il s'avère que ces deux points représentent des objectifs contradictoires. En effet, le fait même d'intercepter un événement et d'enregistrer des informations ralentit et donc perturbe l'exécution du système. Par son intrusion, un traçage trop détaillé peut générer une trace qui ne correspond pas à une exécution typique et reste donc inexploitable.

Un autre point de performances à considérer est la taille de la trace. En effet, les informations à enregistrer sont sauvegardées dans des tampons mémoire, acheminées directement vers un fichier ou sur le réseau. La question qui se pose alors est si ce "canal" d'enregistrement peut tenir le débit et supporter des quantités de traces importantes.

De ces considérations de complétude et de performances ont découlé deux types de traitements pour observer à postériori le comportement d'un système : le profiling et

1. Sous le terme "systèmes de traces" nous rassemblons ici toutes les solutions d'acquisition ou d'exploitation de traces.

le traçage. Le profiling consiste à relever des indicateurs sur le comportement global du système pendant l'exécution. Typiquement ce sont des compteurs ou des variables statistiques qui reflètent le nombre d'appels de fonctions, le nombre et la proportion d'occurrence de certains événements, le temps d'exécution, etc. Le profiling ne fournit pas d'historique d'exécution mais donne des métriques la caractérisant. L'intrusion est bien moindre, la taille de la trace (du profile) est petite mais cela vient au coût d'un niveau d'information limité [2].

Pour minimiser l'intrusion et empêcher le ralentissement du système, les opérations de traçage tendent à être implémentées à un niveau très bas (noyau système) ou même faire appel à des mécanismes matériels. [3].

Comme le projet SoC-TRACE se focalise sur l'exploitation des traces d'exécution i.e sur la phase d'analyse, dans la suite nous considérons les aspects de la phase d'acquisition qui conditionnent l'analyse. Il s'agit notamment du type et de la quantité d'informations qui se retrouvent dans les traces, ainsi que du type et de l'organisation du stockage persistant utilisé.

Une solution d'acquisition de traces fait des choix sur les événements à considérer, décider par quelles informations les caractériser et sous quelle forme les stocker.

### 2.1.1 Choix d'événements à considérer.

Le solutions de traçage peuvent considérer des événements matériels (par exemple relever les compteurs processeur) ou logiciels. Dans le deuxième cas, elles ciblent typiquement une couche logicielle spécifique. Ainsi, il est possible de tracer le système d'exploitation [**?**, **?**], les intergiciels [**?**, **?**, **?**] ou les applications [**?**].

Un aspect qui différencie les outils de traçage est la flexibilité lors de la définition des événements à tracer. Il s'agit de la spécification de l'ensemble des événements qui peuvent être tracés, ainsi que de la décision des événements qui seront effectivement tracés lors de l'exécution. Ainsi, plusieurs solutions se présentent. La plus simple est de ne considérer qu'un ensemble prédéfini d'événements qui sont systématiquement tracés. Une approche plus performante est de pouvoir activer/désactiver la traçage d'événements choisis parmi l'ensemble d'événements prédéfinis. Cette configuration peut être faite avant l'exécution ou dynamiquement. Encore plus flexibles sont les solutions qui admettent la prise en compte de nouveaux événements. Dans le cas des noyaux systèmes, par exemple, les événements sont bien identifiés et peuvent être tous tracés. Ainsi, des solutions comme LTTng [**?**] considèrent un ensemble prédéfini d'événements où l'activation ou la désactivation du traçage peut se faire dynamiquement.

2. Le conflit entre complétude et performances peut être aussi représenté comme un conflit entre une représentation globale/agrégée et la quantité d'information représentée. Plus la représentation est simple (agrégée) plus l'information exacte sur le comportement est perdue. A l'inverse, une représentation détaillée fournit beaucoup d'information mais est difficile à appréhender.

3. Le projet SoC-TRACE s'intéressant à la gestion de traces déjà capturés, les mécanismes particuliers d'instrumentation d'un système et d'acquisition des traces ne seront pas considérés dans la suite de ce rapport

### 2.1.2 Informations sur les événements considérés.

En fonction de l'événement à tracer, le nombre et le type d'informations à relever sont différents. Typiquement, la trace d'un événement contient une indication du moment où l'événement s'est produit (une *estampille*), une information sur le type d'événement, suivis d'un ensemble varié d'informations complémentaires. Le type d'événement peut, par exemple, indiquer qu'il s'agit d'appel de fonction, d'une interruption, d'un envoi de message, etc. Dans le cas d'un appel de fonction, les informations complémentaires pourraient être l'identification de l'appelant, de l'appelé, le nom de la fonction et les valeurs des paramètres.

Dans une grande majorité de cas, les informations de trace des événements sont représentées à l'aide de structures de données plates à plusieurs champs. Ces structures de données se retrouvent directement dans les supports de stockage persistant des traces. Comme la plupart des solutions enregistrent les traces dans des fichiers, il est commun de parler de *format de traces*. Ainsi, la notion de format reflète aussi bien le type des informations de traçage que l'organisation de ces informations au niveau stockage. En d'autres termes, le format confond le modèle de données de traçage avec leur représentation physique.

En ce qui concerne le format d'enregistrement des événements, il est possible de faire la différence entre deux approches. La première, plus simple, est de fixer le type et la structure des enregistrements. Le format de trace obtenu est alors fixe et les fichiers de traces ne contiennent que les données des événements tracés. Le problème avec cette solution est que le format est spécifique au système tracé et ne peut être exploité que par des outils dédiés. La deuxième approche permet de définir la structure des informations se retrouvant dans le fichier de trace (formats auto-descriptifs). Pour ce faire, la trace contient une partie descriptive des données tracées (des méta-données). Les méta-données spécifient la structure des enregistrements et éventuellement donnent des indications sur la sémantique des événements tracés. Les indications de sémantique restent néanmoins assez simples et peuvent typiquement se résumer à un commentaire. A notre connaissance, il n'existe pas d'approche systématique qui permettent la définition et l'exploitation de la sémantique des traces [4]. Par conséquent, pour exploiter les fichiers de traces, qu'il s'agisse de formats fixes ou auto-descriptifs, il est nécessaire d'avoir une connaissance du domaine applicatif (système tracé) et du format particulier utilisé.

### 2.1.3 Stockage des traces.

Le stockage des traces est principalement conditionné par les aspects de performances lors de l'acquisition. En effet, il doit être possible d'enregistrer les flux d'informations de traçage rapidement, sans imposer une perturbation additionnelle au système. Typiquement, les informations de traçage passent par de tampons mémoire avant d'être stockées dans des fichiers. Dans la plupart des cas, les données sont encodées afin de diminuer la taille de stockage. Le résultat de traçage peut être stocké dans un seul fichier ou distribué sur plusieurs fichiers afin de faciliter et accélérer une exploitation ultérieure.

4. C'est d'ailleurs un des objectifs du projet SoC-TRACE

## 2.2 Exploitation de traces

Qu'est-ce qu'on sait faire avec de traces d'exécution ? Considérons l'aspect d'exploitation de traces i.e de travail avec les traces après la phase d'acquisition. La plupart des solutions d'acquisition viennent accompagnées par des librairies fournissant des primitives d'accès aux données enregistrées. Dans le cas le plus simple, il s'agit des accès en lecture et en écriture. Dans d'autres cas, des opérations plus complexes peuvent être fournies comme, par exemple, le filtrage, la calcul de statistiques ou l'agrégation de données. Néanmoins, dans une majorité de cas, ces librairies d'"exploitation" ont pour objectif la représentation visuelle de l'information et non l'extraction de connaissances d'un niveau sémantique plus élevé. Par conséquent, l'extraction d'information utile et la détection d'anomalies dépend majoritairement de l'expérience des développeurs.

Dans la suite, nous énumérons les principaux types de traitements sur des traces d'exécution que nous identifions.

### 2.2.1 Accès en lecture/écriture

Pour capturer et ensuite manipuler des traces, les deux fonctions indispensables sont l'écriture et la lecture de données. En fonction du format de la trace et du domaine d'application, ces primitives sont plus ou moins optimisées en incluant des aspects d'encodage, de manipulation de données binaires, de protection ou de parallélisme. L'objectif principal dans l'implémentation est la performance en temps d'exécution par rapport à la quantité de données manipulées.

### 2.2.2 Filtrage/Sélection

Les traces sont généralement capturées avec un niveau de détail important qui n'est pas nécessaire pour tous les traitements d'analyse. Ainsi, il est courant d'effectuer une sélection d'événements intéressants avant d'extraire des informations d'analyse. La sélection est souvent temporelle comme par exemple ne considérer qu'une période d'exécution (fenêtre) pour la visualisation. La sélection peut aussi être faite sur le type de l'événement comme par exemple s'intéresser aux appels de fonctions.

### 2.2.3 Calcul d'état

Dans beaucoup de solutions, les traces capturées ne contiennent pas de suivi de l'état du système qui est, en réalité, une information métier (information sémantique). Les traces contiennent plutôt des événements ponctuels qui indiquent des changements d'état mais qui ne peuvent être interprétés comme tels qu'uniquement avec un traitement approprié. Par exemple, dans le cas d'applications MPI, il faut relever un événement d'appel de fonction d'envoi de message (MPI_Send) et un événement de réception (MPI_Receive) pour déduire qu'il y a une communication entre des processus.

### 2.2.4 Calcul de statistiques

Les statistiques mettent en perspective l'importance d'un événement par rapport à une durée d'exécution du système. Ainsi, il est parfois intéressant de connaître le nombre d'appels de certaines fonctions, de connaître l'occupation du CPU, de savoir combien de temps certains chemins d'exécution ont pris, etc. Beaucoup de ces informations peuvent être calculées pendant l'exécution comme des fonctions de profiling. Néanmoins, pour minimiser l'intrusion et la charge de calcul pendant le processus d'acquisition de la trace, ce type de statistiques peuvent également être calculées sur les traces une fois l'exécution terminée.

### 2.2.5 Agrégation

L'agrégation consiste à appliquer une fonction à un ensemble d'événements afin d'obtenir un résultat représentant cet ensemble. Cela permet de diminuer le nombre d'événements à considérer et donc de simplifier le travail avec les traces. Néanmoins, l'agrégation masque le niveau de détail et dans la majorité des cas résulte en une perte d'information. C'est typiquement une question critique dans le domaine de la visualisation où représenter tous les événements d'une trace résulte en une visualisation surchargée. La visualisation agrégée de la trace, par contre, peut cacher des situations à problèmes ou amener à de fausses conclusions [**?**].

### 2.2.6 Groupements

Dans beaucoup de traces, des ensembles d'événements sont liés par la logique métier de l'application ou par l'architecture logicielle ou matérielle d'exécution. Ainsi, il est intéressant de considérer tous les événements concernant un thread, tous les threads au sein d'un processus, tous les événements sur un coeur, etc. Pour facilement accéder et manipuler ces groupes d'événements, il est nécessaire de disposer de techniques pour les représenter et les stocker. Dans beaucoup de formats de traces, il est courant, par exemple, de rajouter à l'événement une information sur son producteur i.e sur le processus/le coeur/le thread qui a été responsable de son déclenchement. Ainsi, le format OTF [**?**] implique la génération de traces dans plusieurs fichiers, un fichier représentant un processus.

### 2.2.7 Détection de motifs

La détection de motifs est un traitement qui est très lié à la logique métier du système tracé. L'objectif principal est de détecter des endroits où l'exécution est perturbée ou même erronée. Il s'agit donc de détecter les situations où l'exécution se passe comme prévu et celles où il y a des problèmes. Par exemple, il est courant de détecter les phases de communication dans les applications MPI et de regarder si il n'y a pas d'attentes intempestives ou des blocages.

### 2.2.8 Visualisation

La visualisation a pour objectif de représenter les résultats de tous les traitements énumérés précédemment. En réalité, un grand nombre d'outils [**?**, **?**], présentés comme des outils de visualisation, effectuent des traitements d'analyse de traces avant de représenter cette analyse visuellement.

## 2.3 Synthèse

La majorité des systèmes de traçage et de profiling se concentrent sur l'aspect d'acquisition d'informations en essayant de réconcilier intrusion et niveau de détail. L'analyse de ces informations est laissée à l'appréciation des développeurs expérimentées.

Une partie des systèmes de traçage va plus loin et fournit des traitements qui exploitent la trace et qui fournissent des éléments d'analyse. Il s'agit surtout des environnements et outils développées dans le domaine du HPC. Les raisons principales de cette différence sont deux. Tout d'abord, l'échelle des systèmes d'exécution (nombre de machines, de coeurs, de processus,...) fait qu'il est très difficile d'analyser les traces résultantes. D'autre part, le HPC utilise des standards de programmation qui servent de base pour les différentes analyses.

De point de vue des systèmes embarqués, les systèmes de traçage maîtrisent bien l'aspect d'acquisition de traces par leur utilisation de mécanismes matériels (JTAG) et la limitation respective de l'intrusion. Néanmoins, l'exploitation de traces d'exécution est encore à un stade initial. En effet, les systèmes existant fournissent une visualisation standard "à la Gantt" de l'historique d'exécution et un petit nombre de calculs de statistiques. L'utilisation d'interfaces de programmation propriétaires [5] constitue un frein dans le développement de traitements d'exploitation de la trace plus élaborés comme, par exemple, la détection de motifs. Pour ce faire, il est nécessaire d'inclure en tant que paramètre d'entrée, des informations métier.

De manière générale, les systèmes de traces fournissent un ensemble de traitements d'analyse prédéfini. Par conséquent, une session de travail et d'analyse d'une trace est limitée à l'exploration de la trace brute et à l'examination des quelques éléments calculés par le système de traces. En effet, dans beaucoup de cas, les développeurs se voient contraints à se tourner vers d'autres outils pour effectuer des analyses plus sophistiquées ou tout simplement différentes. Il peut s'agir de scripts ad hoc, de l'utilisation d'outils statistiques, de visualisations différentes, etc. Pour faciliter le travail avec une trace, il est nécessaire de fournir des supports pour mener à bien une session de travail. Pour cela il faut rendre moins rigide le processus d'exploration : il doit être possible de rajouter des traitements d'analyse, de facilement passer d'un traitement d'analyse à un autre et surtout garder la mémoire/l'historique d'exploration et d'analyse de la trace. De manière

5. Les systèmes d'exploitation qui tournent sur des plates-formes embarqués peuvent être considérés relativement similaires, incluant des systèmes Linux et Temps Réel, avec des interfaces "à la POSIX". Toutefois, les intergiciels qui fournissent les services sur lesquels se basent les applications embarqués sont propriétaires.

plus détaillée mais certainement non exhaustive , un système pour l'exploitation de traces doit traiter les points suivants :

- ***Sauvegarder des résultats d'analyse*** L'analyse d'une trace par un développeur peut amener à des conclusions numériques et statistiques ou tout simplement à des idées. Il doit être possible de capturer et sauvegarder ces observations avec la trace analysée. Il peut s'agir de simples annotations, de réflexions ou de structures complexes. Ce point est lié au fait de garder la mémoire de la session d'analyse.
- ***Enrichissement de la trace*** Il doit être possible pour le développeur qui connait le métier de rajouter des informations pertinentes sur la trace. Ces informations pouvant être sauvegardées au même titre que les résultats d'analyse.
- *Sélection spécifique sur les événements de la trace* Un analyste de trace doit pouvoir travailler avec les événements de la trace qu'il considère pertinents. Il doit donc avoir des moyens pour préciser avec quel ensemble il veut travailler. Il doit être facile de combiner des critères prédéfinis (périodes de temps, types d'événements, durées d'exécution, etc.) ver des critères spécifiques programmables par le développeur.
- ***Rajout de nouveau traitements d'analyse*** Comme pour la sélection, un analyste doit avoir la possibilité de spécifier un traitement d'analyse qui lui est propre.
- ***Faciliter la collaboration entre outils*** Si un traitement d'analyse n'est pas immédiatement accessible dans l'environnement d'analyse de traces, il doit être possible de faire collaborer l'environnement et l'outil et même intégrer l'outil.
- ***Programmer l'enchainement des traitements d'analyse*** Une perspective à long terme est de pouvoir programmer (et donc automatiser) des parties de la session d'analyse. Il s'agirait de spécifier le *workflow* et le *dataflow* entre les différents traitements d'analyse, en spécifiant quels résultats doivent être sauvegardés.

# 3 Systèmes de traces existant

Dans cette section nous nous intéressons aux systèmes de traces au sens large. Nous considérons aussi bien les systèmes centrés sur l'acquisition que ceux orientés sur l'analyse et la visualisation. Notre objectif est d'étudier les motivations des systèmes et de les classifier selon leur domaine d'application et les fonctionnalités fournies.

Nous faisons une distinction entre les traitements avancés sur les traces effectués par les outils, d'une part, et le support utilisé pour le stockage et l'accès aux traces, d'autre part. Dans les premiers nous incluons les traitements d'analyse et de visualisation (l'interface) qui sont fournis à l'utilisateur chargé de l'analyse des traces. Ce sont ces traitements que nous considérons dans cette section. Sous le terme support nous rassemblons les aspects liés au contenu des traces, à leur organisation structurelle, au stockage persistant et aux librairies de base fournies pour leur manipulation. Le support est traité dans la section suivante (Section 4).

## 3.1 Systèmes d'exploitation et systèmes embarqués

### 3.1.1 LTTng

Le projet open-source LTTng (Linux Trace Toolkit Next Generation)[**?**] fournit un outil de traçage pour les systèmes Linux. Il considère l'état et les événements du noyau système mais fournit également un mécanisme de traçage pour les programmes utilisateur. Il s'est imposé dans le monde Linux et dans le domaine embarqué grâce à son faible surcoût de traçage.

En ce qui concerne le noyau, LTTng permet de tracer tous les appels de fonctions, ainsi que de récupérer des informations d'état[**?**]. Ainsi, l'outil permet de tracer les interruptions, les appels système, les opérations de gestion de mémoire, l'ordonnancement, etc. En ce qui concerne l'état du système, LTTng peut récupérer la liste des modules noyau chargés, la liste des interfaces réseau, les descripteurs des fichiers ouverts, les informations sur les processus et bien d'autres.

En ce qui concerne le traçage en espace utilisateur, LTTng fournit les mécanismes de définition et d'activation de points de traçage. Contrairement au traçage noyau, où les informations récupérables sont prédéfinies, le choix des points à tracer est laissé au développeur.

LTTng produit des traces au format CTF (Common Trace Format)[**?**].

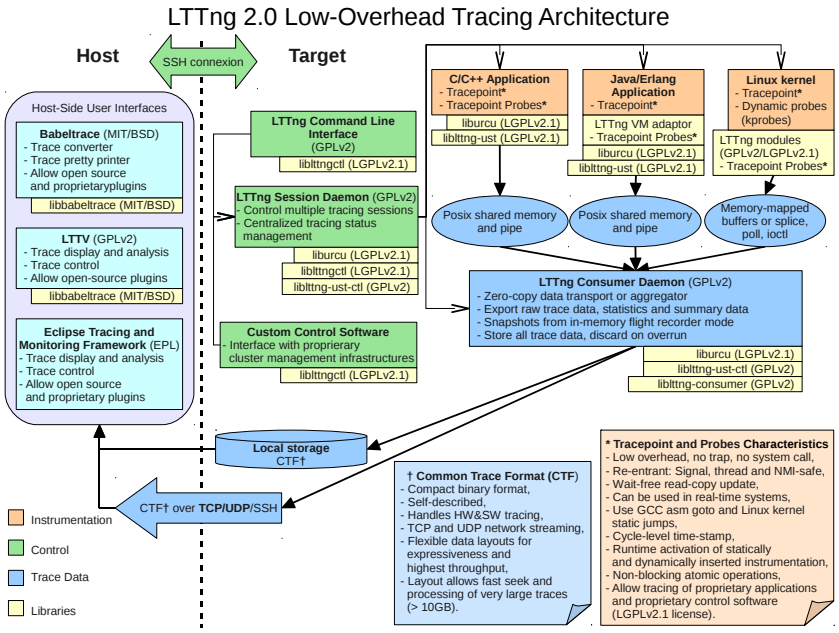


FIGURE 3.1: Architecture de LTTng (source [**?**])

### 3.1.2 STWorkbench

STWorkbench [**?**, **?**] est un environnement complet de développement de STMicroelectronics. Il fournit, entre autres, de outils de débogage, de traçage, de profiling et de visualisation .

En ce qui concerne le traçage, STWorkbench travaille avec KPTrace [**?**] basé sur les distributions STLinux. L'outil active dynamiquement les points de traçage qui concernent principalement des unités de calcul, des flots d'exécution et des interruptions. L'outil peut également observer les symboles du noyau (p. ex. le traitant d'interruptions, les appels du système, les changements de contexte, les fonctions et leurs arguments, etc.), les opérations sur la mémoire (allocation et libération), etc. Le format de trace qui est utilisé est un format propriétaire portant le même nom i.e KPTrace.

Pour le profiling, STWorkbench fournit des profils "plats" standards sur le nombre d'occurrences, le temps d'exécution et l'occupation CPU pour les systèmes STLinux et OS21 (système temps réel de STMicroelectronics). Dans le cas de STLinux, les outils fournis incluent gprof, gcov et surtout OProfile qui permet de considérer l'ensemble des processus et threads s'exécutant sur le système embarqué. Pour OS21, STMicroelectronics fournit un profileur propriétaire.

La visualisation des traces et des profils se fait dans des vues Eclipse qui incluent surtout des diagrammes de Gantt et des vues tabulaires.

### 3.1.3 Intel VTune Amplifier XE

La suite VTune Amplifier XE fournit un ensemble d'outils de profiling et d'analyse de performances pour les plate-formes Intel [**?**, **?**]. Les fonctionnalités fournies par la suite peuvent être qualifiées de standard : échantillonnage statistique de l'utilisation des processeurs, calcul des occurrences d'appels de fonctions, graphe d'appels, calcul des états d'attente, etc. L'avantage des outils de VTune consiste à être adaptés et spécifiques aux architectures des processeurs Intel. de ce fait ils ont accès aux informations pertinentes pour effectuer les différentes analyses de performances de manière efficace. Ainsi, VTune fait usage de la connaissance du mapping des threads sur les processeurs, peut accéder aux compteur matériels spécifiques et avec la dernière version (de la suite et des processeurs) donner des informations sur la consommation électrique.

Avec l'introduction massive du parallélisme au sein des processeurs, Intel a intégré à ces outils de profiling et de monitoring des fonctionnalités liées au multithreading et aux modèles de programmation parallèles utilisés. Ainsi, VTune Amplifier XE analyse las problèmes d'accès concurrents, les attentes, les communications entre threads et processus. Il fournit un support pour TBB [**?**], OpenMP [**?**] et OpenCL[**?**].

## 3.2 Systèmes parallèles

La programmation et l'optimisation d'applications parallèles dans le domaine du calcul haute performance est depuis longtemps confronté aux problèmes de profiling et de traçage. Par conséquent, de nombreuses solutions ont été élaborées pour faire face aux aspects liés à l'acquisition et à l'exploitations de traces. Dans la suite, nous présentons brièvement les outils qui représentent l'état de l'art du domaine.

### 3.2.1 TAU

TAU [**?**, **?**] est un projet de l'université d'Oregon représentant plus de 18 ans de développement. Ciblant l'évaluation de performances d'applications parallèles à travers de profiles et de traces, TAU met un accent fort sur les mécanismes efficaces d'instrumentation et de mesure de systèmes. En mettant en place une panoplie de mécanismes d'instrumentation à la source, au niveau pré-processeur ou compilation, TAU donne accès à un ensemble riche et configurable d'informations. En font partie les événements ponctuels de début ou de fin d'appels, des changements de valeurs, des informations d'intervalle (calcul d'état), des statistiques sur le temps d'exécution de régions de code, etc.

En ce qui concerne les traces d'exécution, TAU permet de les capturer et de les stocker sous forme de fichiers en utilisant son propre format. L'approche du produit est de fournir des convertisseurs de formats afin de se rendre compatible avec les nombreux autres outils d'analyse déjà existant dans le domaine HPC. L'architecture globale sur l'analyse est donnée à la Figure 3.2.

En ce qui concerne les informations de profiles, l'approche de TAU rappelle les motivations du projet SoC-TRACE. En effet, TAU fournit une infrastructure complète de stockage et analyse (cf. Figure 3.3) de profiles. Notamment, les informations communes

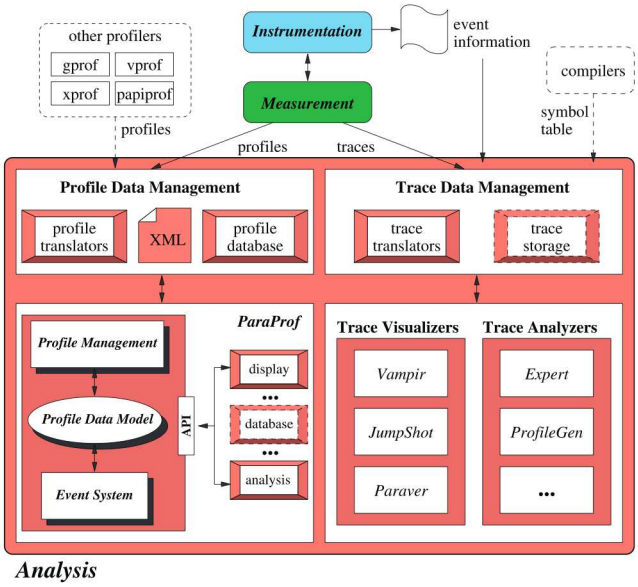


FIGURE 3.2: Analyse de traces et de profiles dans TAU (source [?])

contenues dans les profiles sont représentées en utilisant un format commun, sont stockées dans une base de données et peuvent être manipulées à travers une interface d'accès. Le but étant de pouvoir brancher différentes analyses (fouille de données, analyses statistiques, etc.) sur les profiles.

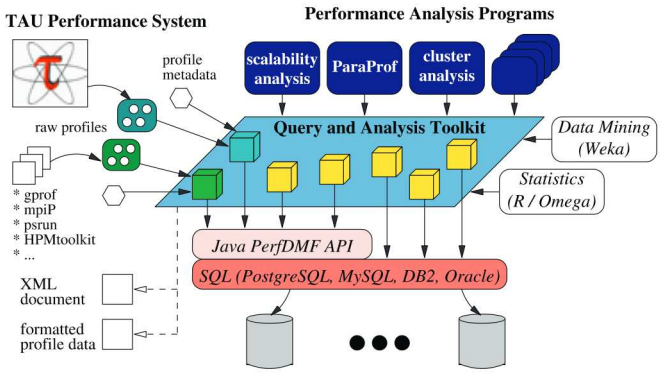


FIGURE 3.3: Infrastructure de gestion de profiles dans TAU (source [?])

### 3.2.2 Scalasca

Scalasca[?, ?] est un outil de profiling et d'analyse de traces pour les applications programmées en OpenMP et MPI. Pour assurer de bonnes performances, Scalasca fait évoluer la version séquentielle KOJAK et implémente l'acquisition et le traitement sur les traces de manière parallèle.

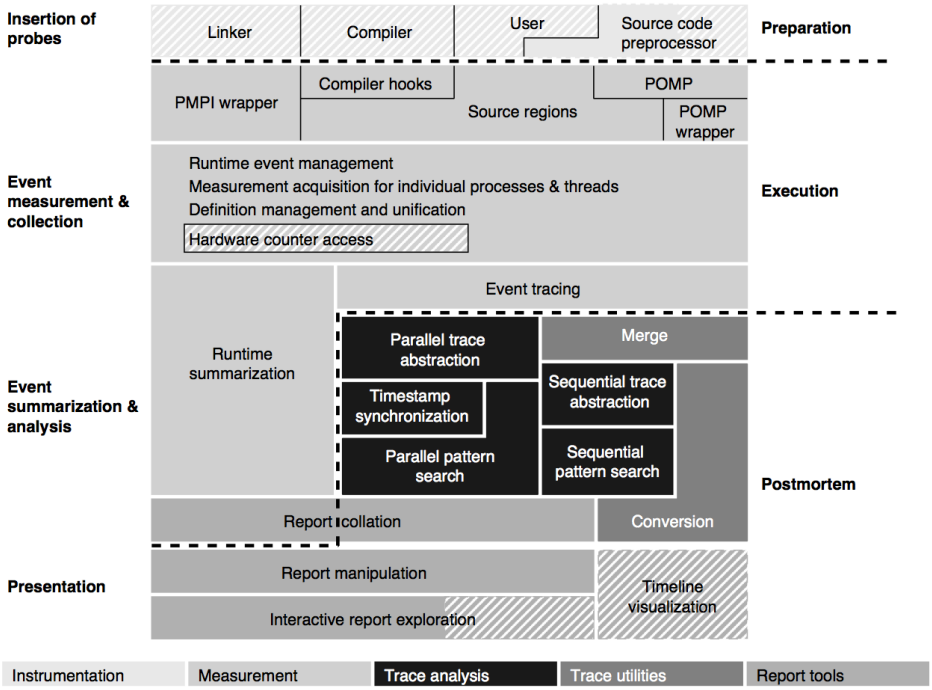


FIGURE 3.4: Fonctionnalités de Scalasca (source [?])

Scalasca fournit une analyse poussée lors de l'exploitation de traces (cf. Figure 3.4). En s'appuyant sur les modèles de programmation et les interfaces bien connus d'OpenMP et de MPI, l'outil intègre des aspects de calcul de statistiques, de filtrage, d'agrégation et de reconnaissance de motifs. La structure des traces reflète la structure en processus et threads de l'application. La reconnaissance de motifs porte majoritairement sur les schémas de communication et de synchronisation correspondant à des problèmes connus par la communauté HPC. Ce point différencie le domaine HPC du domaine embarqué. En effet, du fait de la diversité des applications et du manque de modèles et environnements de programmation standardisés, le domaine embarqué a du mal à bien identifier les situations typiques à problèmes.

Une caractéristique intéressante de Scalasca est également le fait de pouvoir travailler "en session" avec des traces. Les informations de profiling et de traces, ainsi que l'état actuel de la session sont enregistrés dans un répertoire dédié (archive).

Pour résumer, Scalasca est un outil très intéressant par les fonctionnalités qu'il fournit. Néanmoins, ces fonctionnalités sont prédéfinies et il est difficile de rentrer dans la logique

pour en rajouter. L'analyse finale repose sur l'expérience du programmeur.

### 3.2.3 Vampir

Vampir [?, ?] s'adresse à des programmes OpenMP et/ou MPI et est composé de deux parties : VampirTrace pour le profilage et le traçage et Vampir pour la visualisation. VampirTrace permet une instrumentation à différents niveaux et capture des informations standard incluant les graphes d'appels, les historiques d'exécution, les compteurs matériels, l'usage de la mémoire et les E/S. Vampir fournit des vues qui peuvent être considérées standard : vue temporelle, vue de synthèse, vue sur les communications, vue statistiques, etc.

Le point qui distingue Vampir des autres outils dans le domaine HPC est son objectif explicite de supporter l'échelle de très grands systèmes parallèles [1]. Ainsi, VampirTrace adresse les aspects de synchronisation temporelle, de groupement de symboles ( de fonctions, de variables, selon des modules logiciels ou des entités dynamiques) et d'écriture rapide dans de fichiers de traces de format OTF [?].Les traitements d'analyse sont effectués à distance, en gardant les traces "au plus près" du système qui les a produites et en minimisant la quantité d'infirmations à acheminer pour l'analyse. Les traitements d'analyse sont parallélisés et font usage des architectures à mémoire distribuée. Dernier point, VampirServer vise le traitement de traces de très grande taille, de l'ordre de GB.

### 3.2.4 Score-P

La complexité d'analyse des performances des systèmes HPC et la prolifération d'outils de traçage, de profiling et de visualisation sont au coeur des motivations du projet Score-P[?]. Score-P vise la définition d'une infrastructure qui permet l'intégration et la collaboration entre outils existants (cf. Figure). Dans ce sens, Score-P se rapproche aux motivations du projet SoC-TRACE.

Score-P intègre plusieurs outils HPC déjà largement connus : Periscope [?], Scalasca, Vampir et Tau. Score-P travaille avec le format OTF2 pour le traçage et le format CUBE4 pour les profiles.

## 3.3 Autres

Un système différent au sein duquel le traçage est disponible est la machine virtuelle Java. Le traçage permet uniquement de capturer des informations sur l'exécution de la machine virtuelle. Néanmoins, la machine virtuelle ne fournit pas d'outils pour l'analyse de ces informations. Pour analyser l'exécution de Java, il est possible d'utiliser VTune (cf. Section 3.1.3).

1. En Juillet 2013, Vampir a été adapté pour la machine Titan contenant plus de 18 000 GPU et plus de 200 000 CPU. https://www.olcf.ornl.gov/2013/07/22/

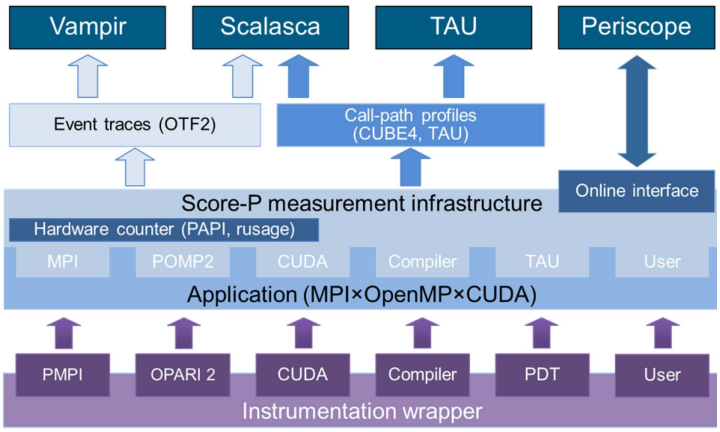


FIGURE 3.5: Architecture de Score-P (source [**?**])

# 4 Formats de traces existant

Dans cette section nous abordons un petit nombre de formats de traces existant, notamment ceux utilisés par les systèmes de traces présentés précédemment. Pour chaque format nous essayons d'expliciter les points suivants :

- Domaine d'application
- Aspects ciblés concernant l'acquisition et l'exploitation de traces
- Moyens de travail avec les traces : interfaces, librairies, outils
- Modèle de données et méta informations
- Connaissances métier

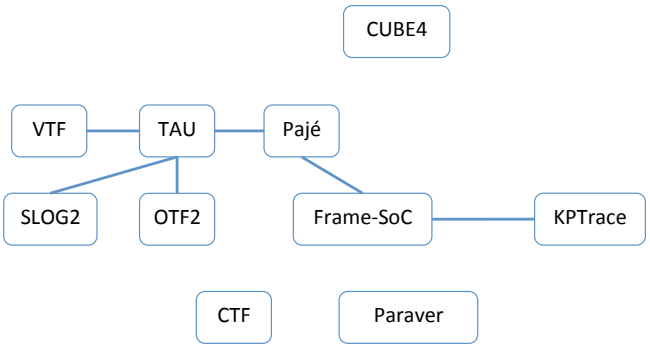


FIGURE 4.1: Conversions existantes entre formats de traces

## 4.1 CTF

CTF (Common Trace Format) [**?**] est le format visant les systèmes d'exploitation et le systèmes embarqués. En particulier, il est produit par LTTng (cf. Section 3.1.1). CTF a été conçu dans le but de produire une trace de taille minimale en un minimum de temps. Les utilitaires fournis pour CTF sont par conséquent plus orientés sur l'acquisition et la relecture de la trace que sur l'analyse des informations capturées. Le support persistant des traces CTF sont les fichiers.

Le format CTF est auto-descriptif et pour ce faire utilise le langage TSDL (Trace Stream Description Language). La structure d'une trace CTF est donnée à la Figure 4.2.

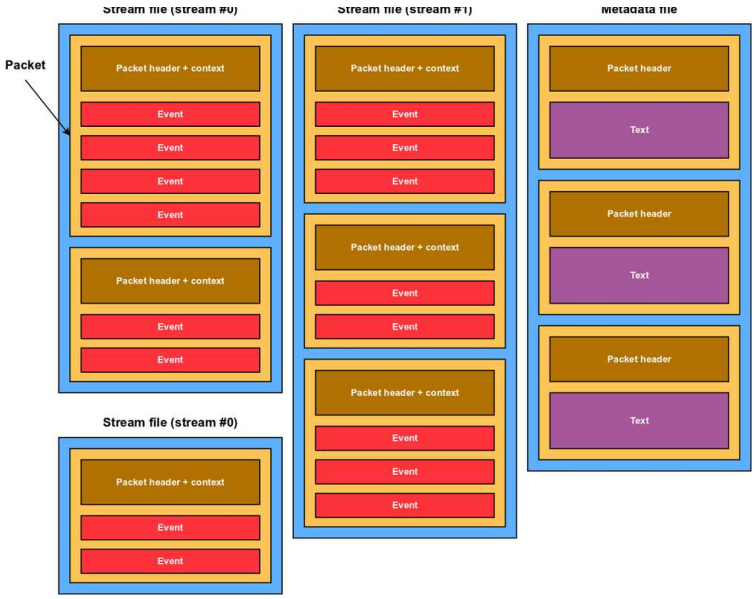


FIGURE 4.2: Structure des fichiers CTF (source [**?**])

CTF trace des événements qui sont organisés en groupes appelés flux (*stream*). Si un flux peut correspondre à un type d'événements ou un contexte d'exécution (par processus, par processeur, etc.), l'utilisation standard qui est faite dans LTTng est de ne définir qu'un flux.

Les enregistrements correspondant aux événements contiennent trois éléments : un identifiant, un type d'événement et des valeurs spécifiques [1]. Les types d'événements sont définis en termes de types de données comme le ferait un langage de programmation. La définition des types inclut la définition d'identifiants uniques et la structuration en termes de types de base (types scalaires, séquences, structures). CTF spécifie également la taille de stockage de ces types, l'alignement, l'encodage ou la compression. Les traces CTF sont par défaut binaires mais il existe également une représentation textuelle JSON [**?**].

Les fonctionnalités de base qui sont fournies avec le format CTF sont des primitives de lecture et d'écriture de la trace. Vu que les fichiers CTF contiennent des définitions (de flux, de types d'événements) et des données réelles, ces primitives sont par conséquent organisés en deux niveaux : lecture/écriture des définitions et ensuite des données. En ce qui concerne la lecture, il est possible de lire toute la trace (lecture séquentielle) ou de lire les événements de manière sélective (accès aléatoire).

L'exploitation des traces CTF peut se faire à travers de deux outils : Babeltrace et TMF. Babeltrace est un outil en ligne de commande qui fournit des primitives de lecture et d'écriture de fichiers de trace CTF. Il permet de convertir les traces CTF dans d'autres formats et principalement permet de retranscrire une trace en un fichier de texte lisible. TMF (Tracing and Monitoring Framework) est un plugin Eclipse qui permet de contrôler

1. De manière similaire aux paquets réseau, cette information est appelée *payload*

LTTng et d'explorer une trace. Les vues fournies incluent une vue détaillée de tous les événements, une vue statistique qui compte le nombre d'événements par type et une vue histogramme qui représente la distribution temporelle des événements.

## 4.2 KPTrace

KPTrace est une solution de traçage produite par ST Microelectronics et s'applique au système STLinux. Le format de traçage est propriétaire et porte le même nom [**?**, **?**].

KPTrace permet de tracer des événements de niveau système incluant les changements de contexte, les interruptions, les appels système, les allocations mémoire, les accès aux structures de synchronisation, etc. Les traces sont enregistrés dans des fichiers ayant une partie d'entête qui décrit les différents symboles utilisés dans la trace brute à la suite du fichier. L'exploitation des traces KPTrace est faite à l'aide des outils de ST Microelectronics comme STWorkbench.

```
Interrupts:
           CPU0
  2:     249718    stx7100_irlm-level     eth0
  5:          0    stx7100_irlm-level     pata_platform
 16:    1199638         stx7100-level     periodic/oneshot time
 18:          0         stx7100-level     TMU2 periodic timer
121:         35         stx7100-level     stasc
140:          0         stx7100-level     fdma_dmac
144:          0         stx7100-level     snd_pcm_player
145:          0         stx7100-level     snd_pcm_player
146:          0         stx7100-level     snd_pcm_reader
147:          0         stx7100-level     snd_spdif_player
168:          1         stx7100-level     ohci_hcd:usb2
169:          0         stx7100-level     ehci_hcd:usb1
End
```

FIGURE 4.3: Partie d'entête KPTrace sur les interruptions (source [**?**])

| Mnemonic | Record format and example | Tracepoint(s) in /etc/kptrace.conf |
|---|---|---|
| C | <timestamp> <PID> C <OLD_PID> <NEW_PID><br>946693049.256575 1403 C 1403 0 | __switch_to |
| I/i | <timestamp> 0 I <IRQ><br><timestamp> 0 i<br>946693049.258883 0 I 16<br>946693049.258981 0 i | handle_simple_irq, handle_level_irq, handle_fasteoi_irq, handle_edge_irq |
| Ix | <timestamp> <PID> Ix<br>946693049.258883 1278 Ix | irq_exit |
| S/s | <timestamp> 0 S <ADDRESS><br><timestamp> 0 s<br>946693049.259021 0 S 84029560<br>946693049.259073 0 s | tasklet_hi_action, net_tx_action, net_rx_action, blk_done_softirq, tasklet_action, run_timer_softirq |
| KC/Kc | <timestamp> <PID> KC <THREAD_FN><br><timestamp> <PID> Kc <NEW_TASK_PID> <NEW_TASK_NAME> | kthread_create |
| KD | <timestamp> <PID> KD <THREAD_NAME> | daemonize |
| KT | <timestamp> <PID> KT <THREAD_FN> | kernel_thread |
| KX | <timestamp> <PID> KX | exit_thread |

FIGURE 4.4: Format KPTrace pour les événements noyau (source [**?**])

KPtrace est exploité au sein d'un système qui fait usage d'une base de données avec

un modèle de données générique. La base de données est accédé et la trace est manipulé au sein de l'environnement Eclipse en fournissant des vues spécifiques. [2]

## 4.3 Tau

TAU génère des fichiers de trace par processeur (tau.trc), ainsi qu'un fichier décrivant les structures d'événements (tau.evt). Dans le fichier de trace, les enregistrements des événements contiennent une estampille, l'identifiant du processeur, du processus/thread, le type de l'événement et des paramètres spécifiques. La trace est distribuée et sous format binaire. L'exploitation se fait en fusionnant d'abord les fichiers de trace et ensuite convertissant la trace dans un autre format (vtf, OTF, etc.).

Le diagramme de conversion entre formats fourni par TAU est le suivant :

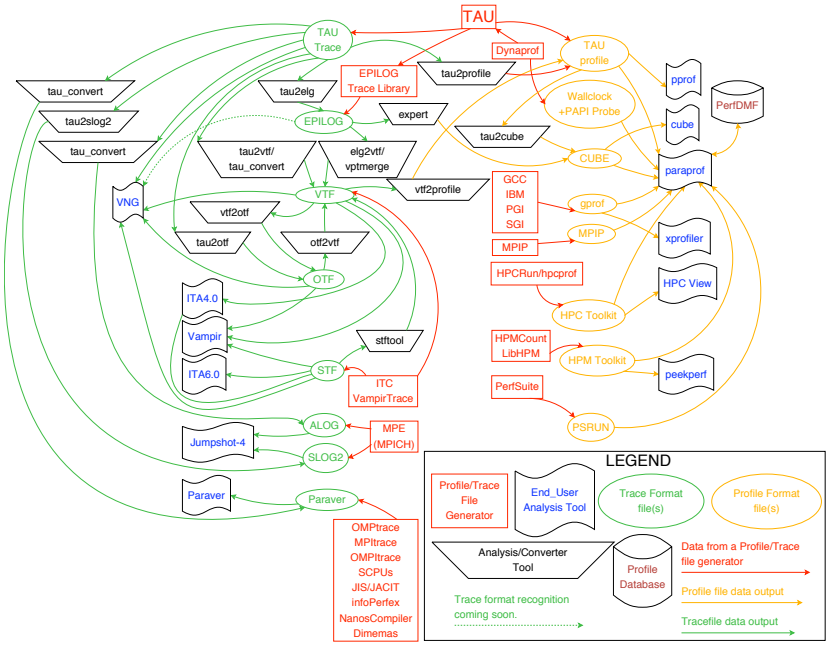


FIGURE 4.5: Exploitation du format TAU (source [?])

## 4.4 OTF2

OTF est un format visant la production et la manipulation de grandes traces sur des machines massivement parallèles. Le support de stockage sont les fichiers et une trace OTF est organisée de manière hiérarchique comme montrée sur la figure suivante.

Une trace est sauvegardée dans un répertoire avec des fichiers multiples. La trace est structurée avec l'aide de flux (*stream*), un flux pouvant typiquement correspondre à

2. Cette logique a été utilisée comme un point de départ pour la conception du prototype pour SoC-TRACE décrit à la suite de ce document.

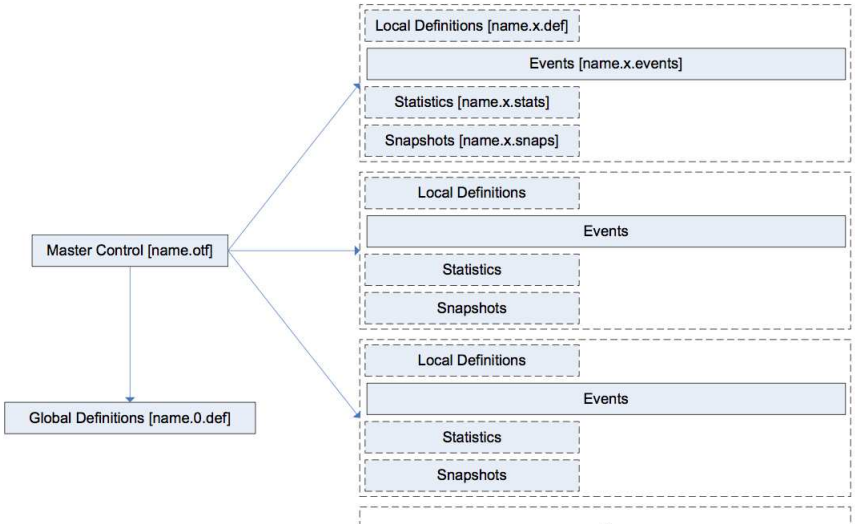


FIGURE 4.6: Structure d'une trace OTF (source [**?**])

un processeur ou un processus. Un flux est décrit en termes de définitions (name.x.def), d'événements (name.x.events), de statistiques (name.x.stats) et de sauvegardes (name.x.snaps). Les définitions concernants des informations globales, comme la résolution des horloges pour la synchronisation temporelle ou le nombre de processus, et des informations locales, comme la structure des événements concernés. Les informations sur les événements sont usuellement regroupées dans un fichier et triés par ordre temporel. Les événements pris en compte concernent les entrées/sorties ddd fonctions, l'envoi/réception de messages, les relevés de compteurs matériels, etc. Les statistiques sont des informations de profiling résumant l'exécution d'une intervalle de temps. Finalement, les sauvegardes permettent de travailler avec une trace sans avoir obligatoirement devoir de lire toutes les informations depuis le début pour reconstituer le contexte d'exécution.

En plus d'une interface de manipulation séquentielle, OTF vient également avec une interface de lecture et d'écriture parallèles de la trace.

## 4.5 CUBE4

Le format CUBE (CUBE Uniform Behavioral Encoding) est utilisé par Scalasca (cf. Section 3.2.2) pour le stockage des données de profiling. Il stocke dans une archive contenant plusieurs fichiers des informations sous forme binaire et fourni une interface pour la création et la lecture de ces fichiers.

CUBE utilise un modèle de données pour les traces d'exécution, appelé *espace de performance*. Cette espace a trois dimensions : la dimension métrique, la dimension de programme et la dimension système. La dimension métrique est composée d'un ensemble de métriques comme, par exemple, le temps de communication ou le nombre de cache miss. La dimension concernant le programme contient l'arbre des appels sur les chemins

duquel il est possible de rajouter des informations métriques. La dimension système référence les entités d'exécution parallèle qui en fonction du modèle de programmation peuvent être des processus, des threads, etc. Chaque point $(m, c, s)$ de l'espace représentera mesure de $m$ au moment où le processus/thread a exécuté $c$. Toutes les dimensions de l'espace sont définies comme des hiérarchies avec des niveaux prédéfinis. Par exemple, la dimension système référence les niveaux machine, nœud, processus et thread.

CUBE fournit une bibliothèque de lecture/écriture de traces organisées selon le modèle décrit. La trace représente une archive de fichiers, faisant usage de XML et de représentations binaires. L'exploitation de traces CUBE se fait de manière visuelle où il est possible d'explorer les données et d'effecter des visualisations par filtrage.

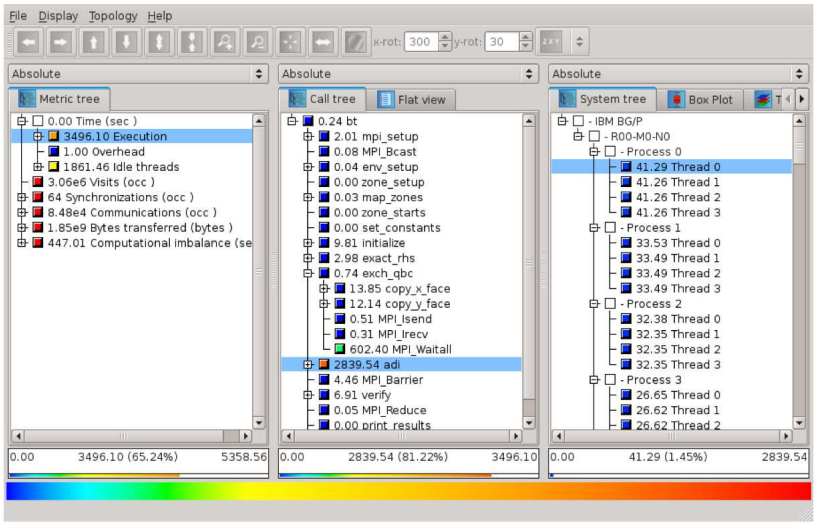


FIGURE 4.7: Visualisation dans CUBE (source [?])

CUBE fournit des mécanismes facilitant l'analyse de multiples expérimentations. Il définit une algèbre [?] avec un ensemble d'opérateurs pour comparer, intégrer et résumer des données CUBE. Cette caractéristique est intéressante et mérite une investigation plus approfondie vis à vis de son applicabilité dans le domaine embarqué.

## 4.6 Pajé

Le format Pajé [?] est un format auto-descriptif sous forme textuelle. Un fichier Pajé contient la définition des types d'événements tracés, la définition d'une hiérarchie de types et une suite d'enregistrements. Les concepts manipulés sont ceux de conteneur, d'état, d'événement, de variable et de lien.

– Conteneur : Un conteneur peut représenter toute entité tracée dont le comportement évolue dans le temps. Un conteneur peut référencer un processeur, un lien réseau, un thread, etc. Un conteneur peut référencer d'autres conteneurs et ainsi définir une hiérarchie dans le système observé.

– Etat : Un état représente un intervalle avec un début et une fin. Un état modélise le fait qu'un conteneur reste dans le même état pendant la période donnée. La définition de la sémantique d'un état est laissé au programmeur.
– Evénement : Un événement tracé, avec une estampille.
– Variable : Une variable suit l'évolution de valeurs numérique pendant le temps. Elle peut représenter la valeur d'un paramètre mesuré ou alors une valeur calculée.
– Lien : Le lien est définit comme une interaction entre deux conteneurs, avec estampille de début et une estampille de fin. Typiquement le lien est utilisé pour modéliser des communications.

Pajè est utilisé par l'outil de visualisation du même nom [**?**, **?**] et ses successeurs Triva [**?**] et VIVA [**?**].

## 4.7 Synthèse

Plusieurs points peuvent être relevés de cet état de l'art sur les formats de traces :

– Support persistant sur des fichiers : La majorité des formats de traces utilisent les fichiers comme support persistant. Cette approche est motivée par les exigences d'efficacité et de vitesse d'écriture, lors de la capture de la trace, et la vitesse de lecture, lors de l'analyse/visualisation de la trace. Par conséquent, un expérimentateur qui aurait à manipuler plusieurs traces de plusieurs exécutions ne peut compter sur d'autres tomes de facilité de travail avec un ensemble de traces que l'arborescence de fichiers existante et une convention de nommage ad hoc.
– Structuration des enregistrements d'observation : Da manière générale, les traces d'exécution contiennent des enregistrements sur les événements et des informations statistiques. Les différents formats de traces définissent de manière différente les structures de données pour enregistrer ces informations. L'approche auto-descriptive fait usage de méta informations pour définir les types, alors que l'approche simple consiste à fournir des types prédéfinis.
  Chaque format introduit ses propres concepts modélisant un système observé. Néanmoins, dans un domaine d'application donné (eg.HPC), les formats utilisés reflètent les mêmes types d'informations et donc les conversions entre formats sont possibles et pratiqués. En d'autres termes, la conversion entre formats est possible grâce à la connaissance métier.
– Méta-données et connaissances sur le système observé : Même si un ensemble de formats de traces contiennent des méta-données décrivant leur contenu, une grande partie des informations sur le domaine d'application, la plate-forme d'exécution ou l'application observée, etc. ne sont pas enregistrées. En d'aitres termes, une trace ne contient pas tout le contexte dans lequel elle a été produite ce qui pose un problème pour une utilisation future.
– Résultats d'analyse : De manière générale, les formats de trace ne sont pas prévu pour enregistrer les résultats des analyses effectuées sur les données de trace. Par conséquent, si une trace doit être consultée plusieurs fois, dans la plupart des cas les analyses doivent être ré-exécutées. Quelques exceptions où des résultats partiels sont

enregistrés sont les snapshots dans OTF/Vampir ou les manipulations de données de profiling dans CUBE/Scalasca.

– Librairies de manipulation : La plupart des formats de traces fournissent des librairies de bas niveau pour la lecture/écriture, avec différents options d'optimisation concernant l'encodage ou la manipulation en parallèle.

# 5 FrameSoC

FrameSoC est notre proposition actuelle d'infrastructure d'exploitation de traces dans le cadre du projet SoC-TRACE. Les points que nous avons considéré lors de sa spécification et développement sont les suivants :

**Non considération de l'intrusion lors de l'acquisition** En effet, SoC-TRACE a pour objectif d'adresser les traitements sur les traces une fois que celles-ci sont capturées par des moyens adaptés, typiquement matériels. L'intrusion n'étant pas un facteur limitant, FrameSoC a pour but de capturer le maximum d'information possible. Pour cela, notre système permet de représenter non seulement les données brutes d'une trace mais également des informations sur l'environnement d'exécution et sur les outils qui ont permis l'acquisition de la trace.

**Gestion de formats hétérogènes** Les formats de traces sont multiples. Un format est typiquement lié à un outil de capture et d'analyse de traces particulier. Il est donc difficile d'utiliser des traces dans le cadre d'un autre environnement d'analyse ou alors cela suppose des conversions de formats qui peuvent induire des pertes d'information. Dans FrameSoC nous proposons un modèle de données générique qui est partagé entre différents outils et qui peut représenter différents formats de traces.

**Représentation d'informations riches en sémantique** Les informations extraites des traces sont présentées de manière visuelle au développeur. Ainsi, même les informations les plus riches de point de vue sémantique ne trouvent leur expression que dans une représentation visuelle qui est plus ou moins fidèle. Ceci est utile pour la détection de situations mais ne permet pas la manipulation et donc une analyse ultérieure des données concernées. Dans FrameSoC nous visons la possibilité de représenter les résultats des analyses des traces et ainsi rendre manipulables des informations complexes. Il peut s'agir de séquences d'exécution, de motifs reconnus, de situations métier spécifiques.

**Travail avec plusieurs traces** Dans beaucoup de systèmes de traces, il est courant de ne travailler qu'avec une trace i.e de n'analyser qu'une exécution à la fois[1]. Dans de rares cas, il est possible de charger dans un outils de visualisation plusieurs traces et de les aligner différemment dans le temps. Le travail d'analyse consiste alors à comparer des sections des traces mais est, de nouveau, laissé à l'utilisateur. Dans FrameSoC, l'idée

1. Nous parlons d'une trace même si la trace d'une exécution peut se traduire par un ensemble de fichiers de trace.

est de pouvoir travailler avec un ensemble de traces et de pouvoir les manipuler et les analyser sans limite.

**Dépôt de traces** L'analyse d'un système nécessite souvent plusieurs exécutions. Or, la gestion des multiples traces qui en résultent est laissée à l'appréciation du développeur et se traduit usuellement en une collection de répertoires avec de nombreux fichiers au sein desquels il est difficile de se repérer. FrameSoC a pour objectif de jouer le rôle d'un dépôt de traces où il est facile de retrouver la trace pour une exécution particulière. Mieux encore, FrameSoC peut servir de mémoire sur l'historique d'exécution et d'analyses effectuées sur un système.

**Stockage et accès à de grosses traces** Les traces d'exécution incluent typiquement des informations de bas niveau comme les valeurs des compteurs matériels, les interruptions, les changements de contexte, etc. La trace d'une courte exécution de l'ordre de quelques secondes peut se traduire en mega- ou giga-octets de données. Pour fournir un stockage persistant et performant pour de telles traces, FrameSoC utilise une base de données relationnelle.

**Facilités d'enchainement des traitements d'analyse** Dans beaucoup de cas il est utile d'effectuer une analyse, de récupérer le résultat et de travailler sur ce résultat avec une autre analyse. Or, les formats de sortie des outils d'analyse de traces sont dans la plupart des formats visuels ou alors non standard. Pour permettre le partage de résultats entre différents traitements d'analyse, FrameSoC propose de sauvegarder les résultats d'analyse en respectant le même modèle de données que les traces.

Dans la suite nous présentons tout d'abord l'architecture générale de FrameSoC , avant de discuter en détail le modèle de données.

## 5.1 Architecture

Figure 5.1 présente l'architecture générale de FrameSoC. La couche inférieure montre la base de données où est représenté le modèle de données générique. Il est important de préciser que le modèle de données est abstrait et donc déconnecté du modèle physique de stockage au sein de la base de données. Le même modèle de données pourrait être implémenté et stocké sur un autre support de stockage comme par exemple les fichiers ou les bases de données non relationnelles.

FrameSoC fournit une couche logicielle qui sert d'interface entre les outils d'analyse et le support de stockage. Dans cette bibliothèque se trouvent toutes les fonctions de base qui facilitent l'accès et la manipulation de données stockées au sein de la base. Actuellement, la bibliothèque contient des fonctions de lecture, d'écriture et de sélection d'événements.

FrameSoC doit jouer le rôle d'un cadre dans lequel peuvent être intégrés différents outils d'analyse et de manipulation de traces. Actuellement, FrameSoC vient avec une interface simple, ainsi que des outils de base d'agrégation et de calcul statistique.

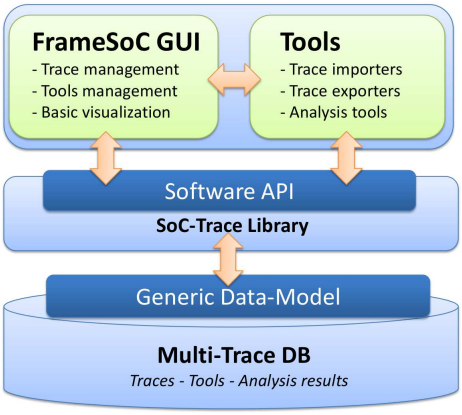


FIGURE 5.1: Architecture de FrameSoC

## 5.2 Modèle conceptuel de données

Le modèle de données est présenté sur la Figure 5.2. Le modèle sépare les données sauvegardées en trois parties : les données contenues dans la trace (*Raw Trace Data*), les informations sur l'acquisition de la trace (*Trace General Information*) et les résultats d'analyse (*Analysis Data*).

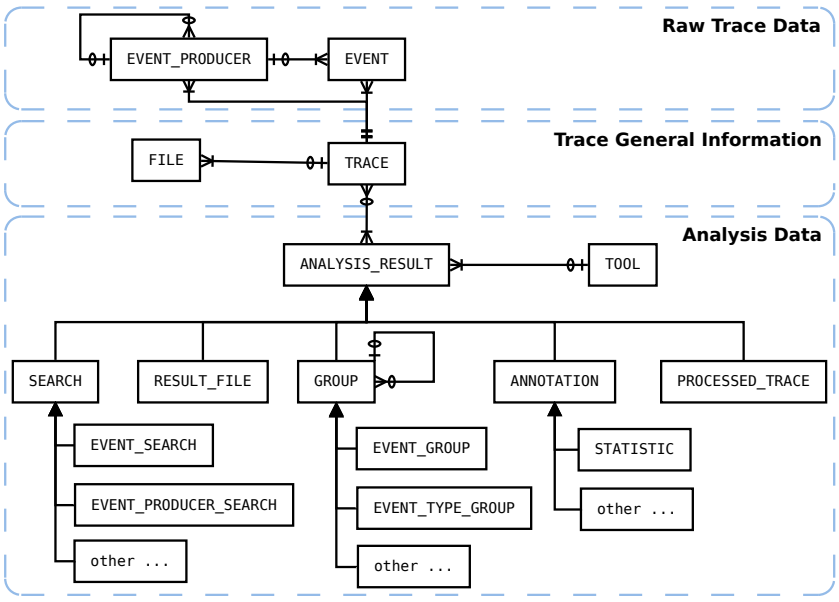


FIGURE 5.2: Modèle conceptuel de données

### 5.2.1 Données de configuration

L'entité qui représente une trace est (`TRACE`). Elle a pour rôle d'enregistrer les conditions expérimentales dans lesquelles la trace a été produite. Typiquement ce sont des informations sur le logiciel qui est tracé (nom, version,...), l'environnement matériel utilisé pour l'exécution (famille et version de *board*, nombre de processeurs, etc), ainsi que toute autre renseignement pertinent ( compilateur, options de compilation, niveau d'optimisation, bibliothèques utilisées, etc). Les données de configuration, ainsi que les données de traçages proviennent possiblement de fichiers dont la liste est référencée (`FILE`)

Pour pouvoir représenter différentes informations sur une trace, l'entité `TRACE` est modélisée en utilisant des entités auto-descriptives (*Self Defining Pattern.* Pour cela, sont utilisées quatre entités : `TRACE`, `TRACE_PARAM`, `TRACE_TYPE` et `TRACE_PARAM_TYPE`. Chaque entité `TRACE` est caractérisée par un type `TRACE_TYPE`. L'entité `TRACE` a des attributs prédéfinis qui donnent l'estampille, le logiciel tracé, la plate-forme matérielle, le nombre de processeurs, le dispositif de traçage et une description. ces attributs ont été choisis pour leur grande fréquence d'usage lors de l'analyse de traces. Le fait de les inclure directement dans l'entité `TRACE` les rend plus facilement accessibles ce qui a un impact sur les performances. Le `TRACE_TYPE` peut être vu comme une extension de description de la trace où sont données des informations complémentaires. Ainsi, chaque `TRACE_TYPE` référence plusieurs `TRACE_PARAM_TYPE` où il spécifie les différents types d'informations qui peuvent être attachées à une trace. Les valeurs spécifiques de ces paramètres pour une trace donnée vont se retrouver dans des entités `TRACE_PARAM`. De point de vue sémantique, travailler avec ses informations demande de prendre connaissance avec leur définition. De point de vue des performances, la manipulation de ces données demande logiquement une indirection et est donc plus lente.

Une trace est également caractérisée par un booléen qui dit si c'est une trace brute (résultat d'uns acquisition) ou une trace déjà traitée. En effet, rien n'empêche de considérer une trace qui a été modifiée (filtrée, rajout d'événements, etc) comme état une trace à part entière qu'il est possible d'analyser et enrichir/modifier à nouveau.

### 5.2.2 Données de trace

Une trace (`TRACE`) est composée d'événements (`EVENT`), un événement étant déclenché par un producteur (`EVENT_PRODUCER`). Exemples de producteurs d'événements sont typiquement les processus ou les threads. Les producteurs peuvent être organisés en hiérarchie (e.g. groupes de processus où un processus est un groupe de threads). Un événement est caractérisé donc par un identifiant unique, une estampille, une information de type et l'identité de son producteur. Dans l'implémentation actuelle de FrameSoC un événement est également caractérisé par un numéro de CPU et un numéro de page. En effet, la première information est très souvent utilisée pour les groupements des événements et leur analyse. La deuxième information est liée aux performances de manipulation d'un grand nombre d'événements où il est impossible de tous les charger en mémoire. Ainsi, typiquement lors d'une visualisation, une seule partie (page) est chargée.

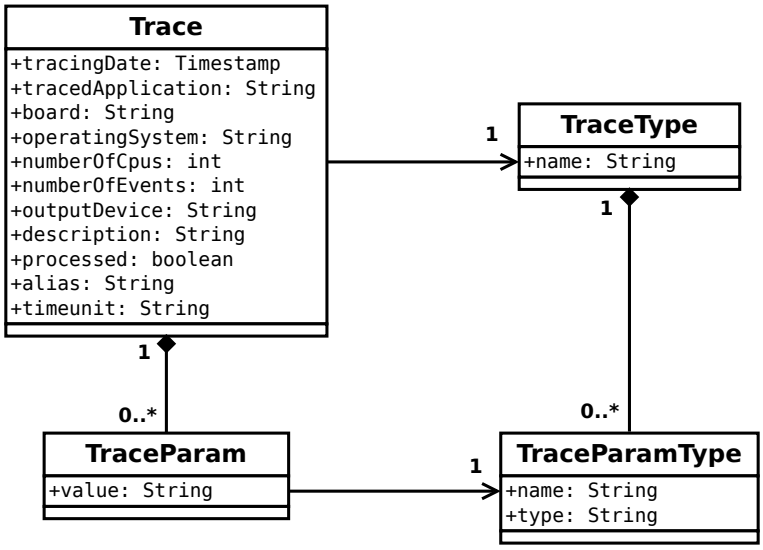


FIGURE 5.3: Modèle de données pour les traces

La définition d'un événement fait également usage du modèle auto-descriptif (Figure 5.4).

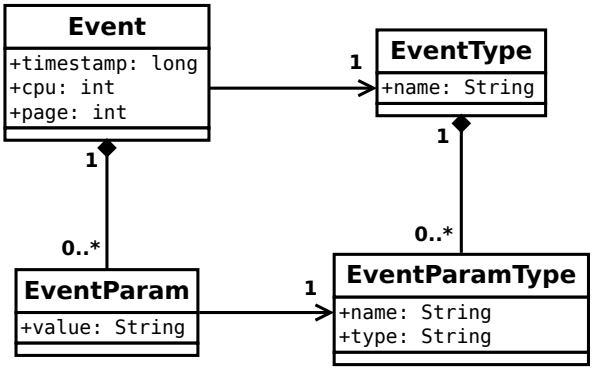


FIGURE 5.4: Modèle de données pour les événements

### 5.2.3 Résultats d'analyse

Pour sauvegarder les résultats d'analyse, FrameSoC s'est inspiré des cas d'utilisation et des besoins des partenaires de SoC-TRACE. Ainsi, quatre types de résultats sont définis : les annotations, les groupes, les filtres et les résultats spécifiques. Les annotations donnent la possibilité de rajouter des informations sur une trace. Les groupes permettent de distinguer des ensembles d'événements (séquences typiques, motifs) et de les organiser en hiérarchie. Les filtres sauvegardent des ensembles d'événements importants selon certains critères. Enfin, les résultats spécifiques sont ceux qui ne rentrent dans aucune

des catégories précédentes.

### 5.2.4 Evolution du modèle de données en décembre 2013

Les travaux de collaboration avec les partenaires, ainsi que les travaux d'Alexis Martin, en thèse depuis septembre 2013, nous ont amené à enrichir le modèle de données afin de catégoriser les informations d'une trace. La notion de *catégorie* rajoute de la sémantique et nous la croyons utile tant de point de vue des analyses de traces, que de point de vue des performances des traitements.

Les catégories que nous rajoutons sont des concepts mis en avant dans de nombreux outils d'analyse et de visualisation de traces et en particulier dans le format Pajé (cf. 4.6). Ainsi, dans la version non enrichie du format de données FrameSoC un événement est un événement ponctuel i.e quelquechose qui se produit à un moment donné dans l'exécution d'un système. Dans la version enrichie du modèle, chaque événement est catégorisé en événement ponctuel, état, variable ou lien. L'entité `EVENT` a trois nouveaux attributs qui sont la catégorie et deux paramètres qui caractérisent cette dernière. Les nouvelles notions sont par conséquent les suivantes :

- *Etat* Un état représente un intervalle avec un début et une fin. Peuvent être modélisés comme des états des phases de calcul, les exécutions des fonctions, etc. Les deux paramètres contiennent respectivement la fin de l'intervalle (le début étant enregistré dans l'estampille) et un niveau d'imbrication.
- *Variable* Une variable suit l'évolution de valeurs numérique pendant le temps. Elle peut représenter la valeur d'un paramètre mesuré ou alors une valeur calculée. Les deux paramètres contiennent l'identifiant de la variable est sa nouvelle valeur.
- *Lien* Un lien représente une relation entre deux entités. La relation peut être une relation causale, une relation de communication ou autre. La première entité est la source de l'événement actuel, avec une datation de la relation dans l'estampille de l'événement. La deuxième entité et la fin de la relation sont encodées dans les deux paramètres pour cette catégorie.

## 5.3 Modèle de données

Le modèle actuel de représentation au sein de la base de données est donné sur la Figure 5.5. La représentation est séparée en plusieurs bases de données : une qui contient les informations relatives à l'environnement de traçage et aux outils (`System_DB`) et une base de données par trace importée (`Trace_DB`).

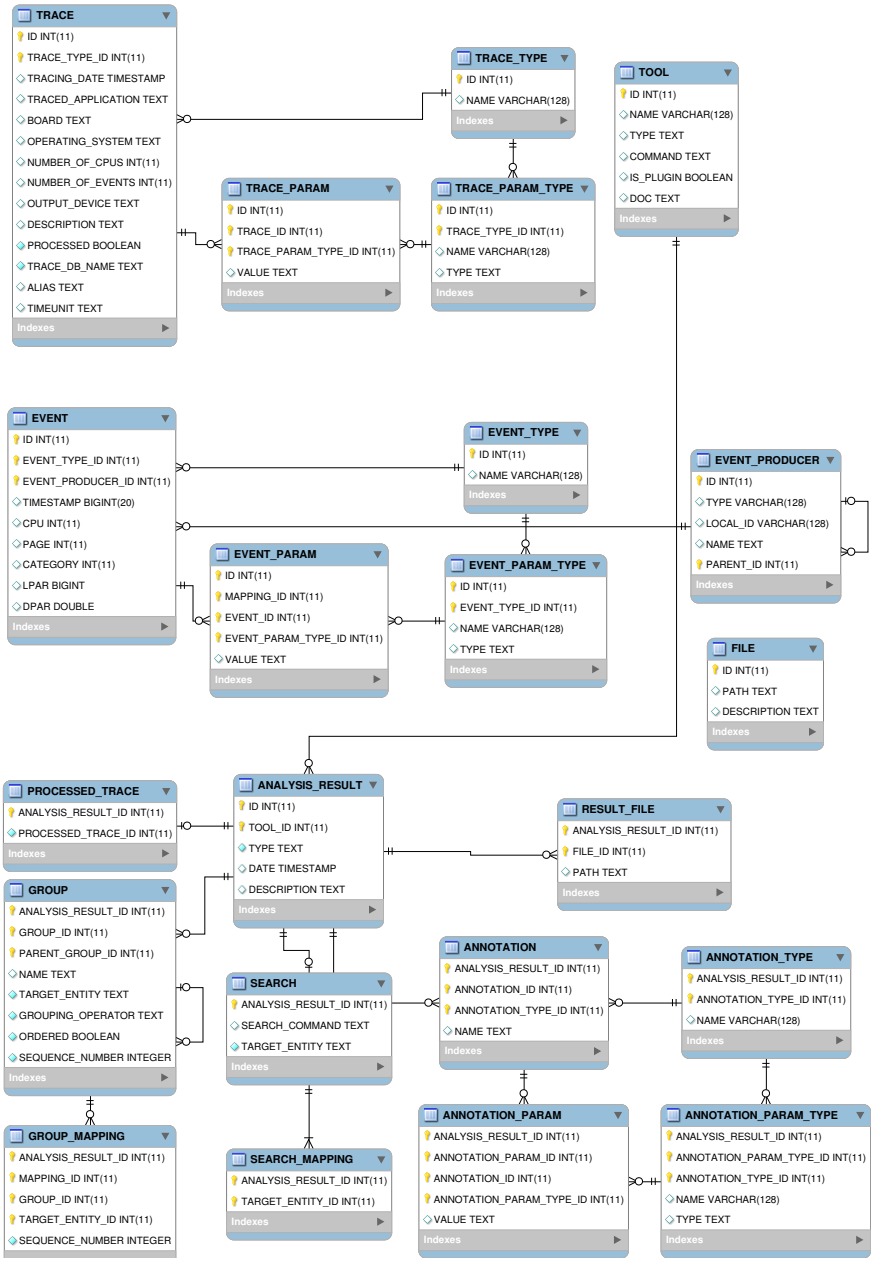


FIGURE 5.5: Modèle de données relationnel SoC-TRACE

## 5.4 Un exemple simple d'instanciation

L'exemple qui suit considère le modèle de données avant son enrichissement avec des catégories.

Considérons l'exemple simple d'importation d'un type d'événement au sein de la base de données de traces. Le type d'événement correspond à un changement de contexte (CS = Context Switch) et est tracé avec le format simple suivant :

```
<timestamp> CS <oldpid> <newpid>
```

Un événement concret faisant partie d'une trace capturée pourrait être :

```
227537525 CS 0 21
```

En termes de modèle de données conceptuel, cela donne 5.6.

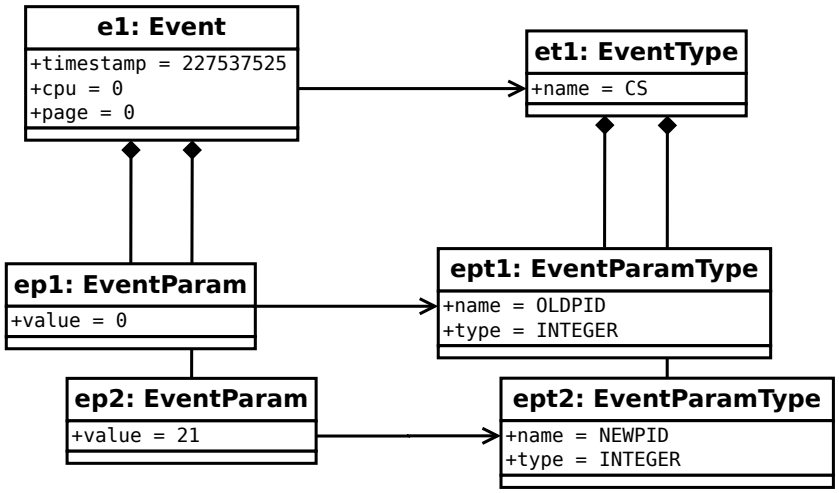


FIGURE 5.6: Modèle conceptuel de représentation d'un événement de changement de contexte (CS)

L'importation dans la base de données se traduira par la structure donnée dans la Figure 5.7.

## 5.5 Validation

Le prototype FrameSoC a été développé pour la fin 2012 et depuis évolue à travers d'interactions au sein du projet SoC-TRACE. Les différentes actions et résultats ont été publiés dans [**?**, **?**, **?**, **?**].

Les principaux résultats d'utilisation en 2013 sont résumés dans ce qui suit.

| EVENT_TYPE | |
|---|---|
| ID | NAME |
| .. | .. |
| 3 | CS |
| .. | .. |

| EVENT_PARAM_TYPE | | | |
|---|---|---|---|
| ID | EVENT_TYPE_ID | NAME | TYPE |
| .. | .. | .. | .. |
| 14 | 3 | OLDPID | INTEGER |
| 15 | 3 | NEWPID | INTEGER |
| .. | .. | .. | .. |

| EVENT | | | | | |
|---|---|---|---|---|---|
| ID | EVENT_TYPE_ID | EVENT_PRODUCER_ID | TIMESTAMP | CPU | PAGE |
| .. | .. | | .. | .. | .. |
| 9 | 3 | 54 | 227.537525 | 0 | 0 |
| .. | .. | | .. | .. | .. |

| EVENT_PARAM | | | |
|---|---|---|---|
| ID | EVENT_ID | EVENT_PARAM_TYPE_ID | VALUE |
| .. | .. | .. | .. |
| 37 | 9 | 14 | 0 |
| 38 | 9 | 15 | 21 |
| .. | .. | .. | .. |

FIGURE 5.7: Importation d'un événement au sein de la base de données

### 5.5.1 Gestion de plusieurs formats

FrameSoC vient avec plusieurs outils d'importation permettant de convertir des traces et de les intégrer au sein de la base de données. Il s'agit de :

- KPTrace : format propriétaire de ST Microelectronics
- Pajé
- Format utilisé par le partenaire TIMA pour l'analyse des accès mémoire.
- GStreamer [?]
- Format "maison" simple utilisé à des fins de test.

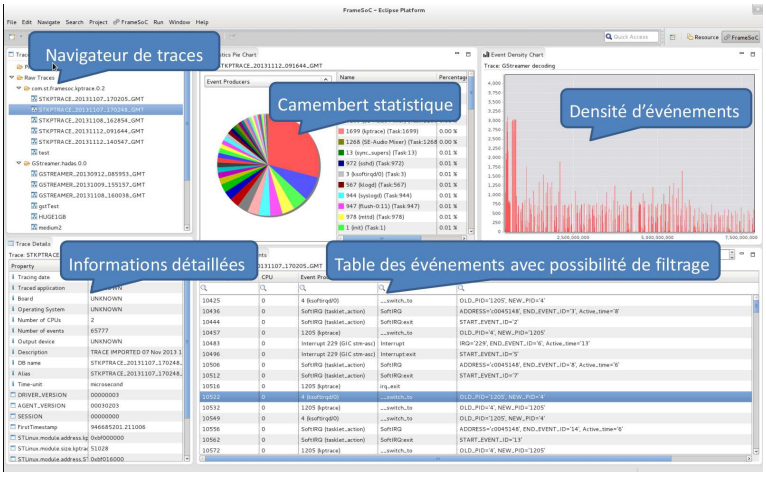


FIGURE 5.8: Vues proposées dans FrameSoC

### 5.5.2 Intégration avec les partenaires de SoC-TRACE

L'interface graphique de FrameSoC est montrée à la Figure 5.8.

L'ensemble de briques énumérées ci-dessous va être utilisé comme base pour la production de la première version du système d'exploitation de traces SET1.0, livrable prévu pour juin 2014.

**Librairie de visualisation TChartsLite et outil de visualisation KPTrace / ST** FrameSoC a intégré l'importateur de traces KPTrace ce qui permet de travailler avec des traces d'exécution qui correspondent aux cas d'usage réels considérés par le projet SoC TRACE. En termes de visualisation, FrameSoC a intégré la librairie de visualisation TChartsLite qui a été utilisée pour l'amélioration des vues proposées. FrameSoC a également intégré des vues spécifiques (Gantt et vues statistiques) pour le format KPTrace.

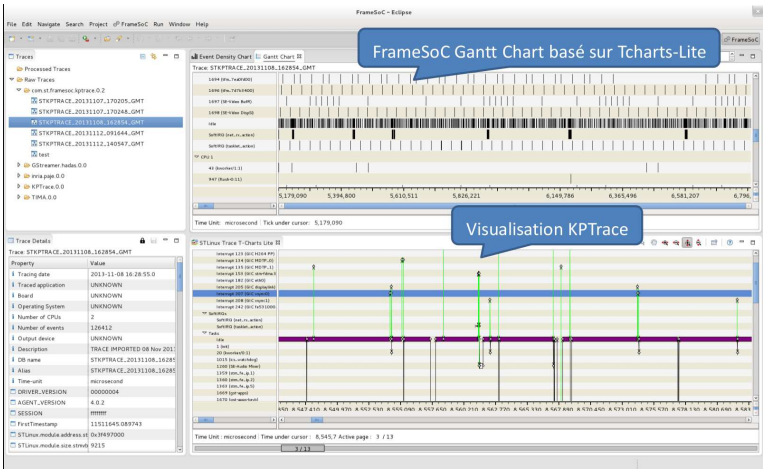


FIGURE 5.9: Vues utilisant les travaux ST

**Visualisation agrégée / INRIA** Il a été possible de connecter le prototype FrameSoC avec l'outil Ocelotl afin de fournir une visualisation agrégée d'une trace stockée au sein de l'infrastructure. Les phases identifiées par cette visualisation peuvent être enregistrées ensuite au sein de FrameSoC. L'algorithme utilisé par Ocelotl est un algorithme d'agrégation de l'information. Au sein de FrameSoC, il a été porté afin d'utiliser les interfaces fournies de manipulation de données. Pour des questions de performances, Ocelotl a effectué des optimisations en termes de requêtes JDBC et de cache applicatif (chargement partiel des données en mémoire). Ocelotl dispose d'une vue Eclipse spécifique correspondant.

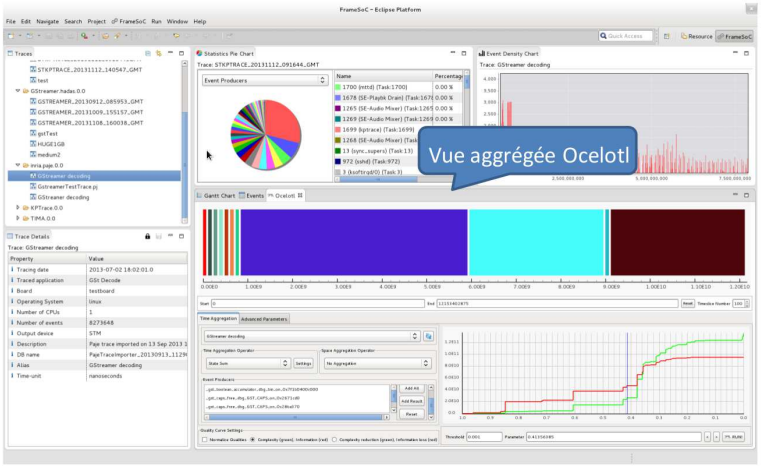


FIGURE 5.10: Vue Ocelotl

**Structuration de traces / LIG** Plusieurs travaux d'intégration ont été menés conjointement avec le partenaire LIG/HADAS. Ont été intégrés les outils Abstractor, ProfSpan et FrameMiner. L'outil Abstractor permet la manipulation de concepts métier à base d'ontologie. Ce travail va se poursuivre dans les mois à venir, l'ontologie ayant pour vocation d'évoluer avec les connaissances des scénarii réels fournis par ST. L'outil ProfSpan permet la recherche de motifs fréquents L'outil FrameMiner permet la recherche des motifs les plus couvrant. Dans les trois cas, les outils ont utilisé l'interface générique d'accès aux données et ont sauvegardé leurs résultats (motifs et concepts métier) en utilisant les types de résultats prévus dans FrameSoC. Les besoins qui ont été identifiés lors de ces travaux sont de pouvoir partitionner la trace, d'effectuer un ensemble de traitements statistiques de base sur ces portions et de sélectionner des portions de la trace.

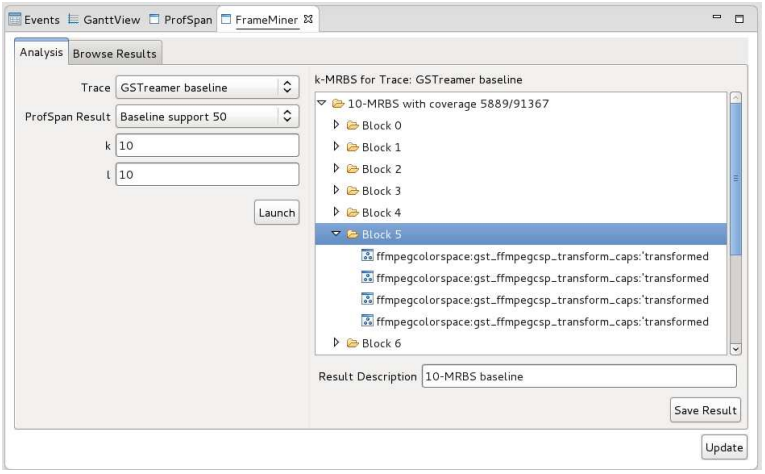


FIGURE 5.11: Vue FrameMiner

**Structuration de traces par analyse probabiliste / ProbaYES** FramSoC a intégré l'outil MegaLog qui permet de prendre en entrée des patterns et ensuite trouver les occurrences et calculer des stats par rapport à toute la trace. L'intégration de l'outil MegaLog a été faite, de la même manière que les outils de structuration, en utilisant l'interface générique pour la lecture des données et la sauvegarde de résultats. La lecture a concerné les motifs produits par l'outil FrameMiner et les résultats ont consisté à sauvegarder des statistiques sur leurs occurrences.

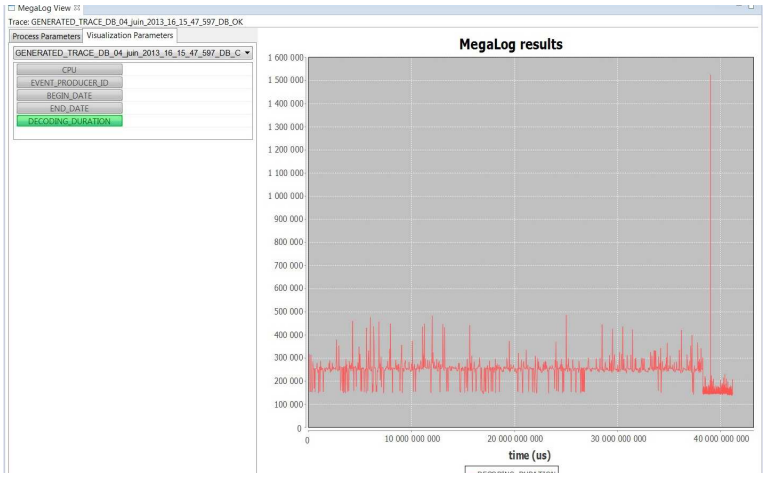


FIGURE 5.12: Vue MegaLog

### 5.5.3 Performances

Le développement du framework FrameSoC a été accompagné de manière continuelle par des tests de performances.

- Temps d'importation :
  Nous avons mesuré le temps d'importation de traces au sein de la base de données FrameSoC. Ce temps doit être acceptable dans un cycle d'analyse de trace. Les résultats ont montré que ce temps peut être important pour de grosses traces, contenant de nombreux événements ayant des structures complexes et diverses. Ainsi, l'importation d'une trace de l'ordre de $3GB$ a pris $8min$ environ. L'importation d'une trace de $10GB$ a atteint près d'une heure. Néanmoins, le temps d'importation de traces utilisées habituellement qui font quelques MB ne prend que quelques secondes. Nous avons testé l'activation de l'indexation de la base de données qui s'est avéré nuisible amenant un ralentissement de près de 70%.
- Temps de retrait de résultats :
  Des tests ont été effectués pour montrer l'intérêt de stocker des résultats d'analyse afin d'éviter de refaire des analyses coûteuses. Des expérimentations initiales sur des calculs de statistiques ont montré que si il fait 3 min de calcul pour une trace de 3GB, le retrait de résultats ne prend que 100ms. Ces expérimentations vont être poussées

plus loin, maintenant que FrameSoC est intégré avec des analyses plus complexes fournies par les partenaires LOG/HADAS et INRIA/MOAIS.

– Temps de filtrage
  Le temps de récupérer des événements "intéressants" dépend du paramétrage de la base de données. Avec indexation, le temps ne dépend pas de la taille de la trace mais de la taille du résultat. Sans indexation, le temps est linéaire en fonction de la taille de la trace.

Les performances restent la grande question ouverte. Il nous manque une utilisation intensive et des use case réalistes. Des expérimentations sont en cours sur le représentation des groupes et l'utilisation de cache logiciel "configurable" pour les besoins spécifiques des outils.

# 6 Conclusion

Dans ce rapport nous nous sommes intéressés aux questions ouvertes liées à l'exploitation des traces d'exécution de systèmes embarqués. Nous avons brossé un état de l'art des systèmes de traçage provenant de différents domaines d'application afin de catégoriser les besoins en termes d'acquisition et d'exploitation de traces. Nos conclusions majeures sont les suivants :

- *Accent sur l'acquisition* : la plupart des outils et des techniques existant se concentrent sur l'acquisition de la trace et sur les aspects d'intrusion et non sur son exploitation ultérieure.
- *Analyse facilitée si interfaces de programmation explicites* : L'exploitation d'une trace capturée reste assez limitée sauf dans les cas, comme dans le domaine du HPC, où le modèle de programmation et les interfaces utilisées sont explicites et bien connues.
- *Le résultat d'analyse se traduit en visualisation* : Même quand les traitements d'analyse sont complexes et incluent des analyses de graphes d'appels, des calculs d'état et la reconnaissance de situations typiques, les résultats de ces analyses trouvent leur expression uniquement dans la visualisation. Ils ne sont pas manipulables autrement.
- *Peu d'interactivité* : Les outils de visualisation de traces sont assez rigides et offrent peu d'intéractivité à l'utilisateur.
- *Spécificité des traitements, formats, visualisation* : La majorité des systèmes de traces viennent avec leur propre format de traces, des traitements propriétaires, des visualisations spécifiques. Néanmoins, leurs fonctionnalités , au moins par domaine, sont très proches. Le besoin d'unification et de standardisation émerge et voit son expression dans des projets comme Score-P ou SoC-TRACE.

FrameSoC, notre premier prototype d'infrastructure de gestion de traces adresse partiellement ces aspects. Ses objectifs principaux sont de passer outre la spécificité des formats de traces, de faciliter l'intégration de différents traitements d'analyse et de permettre la manipulation avancée de résultats d'analyse. Pour cela, il se base sur une base de données et fournit un modèle générique de données permettant de capturer des informations sur les traces brutes, l'environnement de traçage et les résultats d'analyse. Suite à ces travaux, plusieurs questions restent ouvertes :

- *Généricité du modèle de données* : Le modèle de données proposé suffit-il à refléter tous les concepts nécessaires à l'analyse de comportement de systèmes embarqués ?
- *Flexibilité contre performances* : Les approches génériques viennent avec un coût d'utilisation et de fonctionnement supérieur. Quel est le compromis acceptable ?
- *Scénarii d'usage* : La validation de FrameSoC et des outils développés par les partenaires dans SoC-TRACE ne peut se faire sans scénario d'usage typiques du domaine. Quatre cas d'usage ont été présentés par ST Microelectronics en septembre 2013 et

seront utilisés comme base pour la production de la première version du système d'exploitation de traces SET1.0.

– *Liens à d'autres domaines* : FrameSoC peut-il s'inspirer et intégrer des travaux d'autres domaines ? Les concepts et mécanismes de FrameSoC peuvent-ils être appliqués à d'autres domaines ?