Le web sémantique en aide à l'analyste de traces d'exécution

Léon Constantin Fopa, Fabrice Jouanot, Alexandre Termier, Maurice Tchuenté

Le web sémantique en aide à l'analyste de traces d'exécution

Léon Constantion Fopa
LIG, Université de Grenoble
Grenoble, France
leon-constantin.fopa@imag.fr

Fabrice Jouanot
LIG, Université de Grenoble
Grenoble, France
fabrice.jouanot@imag.fr

Alexandre Termier
INRIA - IRISA
Université Rennes 1
Rennes, France
alexandre.termier@irisa.fr

Maurice Tchuenté
IRD-UMMISCO et LIRIMA
Cameroun
maurice.tchuente@lirima.org

ABSTRACT

L'analyse de traces d'exécution est devenue l'outil privilégié pour débugger et optimiser le code des applications sur les systèmes embarqués. Ces systèmes ont des architectures complexes basées sur des composants intégrés appelés SoC (System-on-Chip). Le travail de l'analyste (souvent, un développeur d'application) devient un véritable challenge car les traces produites par ces systèmes sont de très grande taille et les évènements qu'ils contiennent sont de bas niveau. Nous proposons d'aider ce travail d'analyse en utilisant des outils de gestion des connaissances pour faciliter l'exploration de la trace. Nous proposons une ontologie du domaine qui décrit les principaux concepts et contraintes pour l'analyse de traces issues de SoC. Cette ontologie reprend les paradigmes d'ontologie légère pour supporter le passage à l'échelle de la gestion des connaissances. Elle utilise des technologies de "triple store" RDF pour son exploitation à l'aide de requêtes déclaratives SPARQL. Nous illustrons notre approche en offrant une analyse de meilleure qualité des traces d'un cas d'utilisation réel.

Keywords

ontologie, RDF, traces, web sémantique

1. INTRODUCTION ET CONTEXTE

Les systèmes embarqués sont basés sur des System-on-Chip (SoC) qui intègrent sur une même puce plusieurs composants : processeurs, mémoires, port d'entrée/sortie. Ces SoCs sont par exemple utilisés pour exécuter les applications multimédia embaquées dans nos smartphones, tablettes, téléviseurs ou set-top boxes. Les applications exploitant les SoCs sont complexes et peuvent difficilement être déboguées ou optimisées avec les méthodes d'analyse de codes. Les développeurs ont donc recourt à la capture et à l'analyse post-mortem des traces d'exécution [3, 4, 8]. Une trace d'exécution contient des évènements de bas niveau tels que les interruptions, les changements de contexte ou les appels de fonctions. Identifier des problèmes à partir de ces évènements présente deux difficultés. Premièrement la taille de la trace peut atteindre plusieurs millions d'évènements pour seulement quelques minutes d'exécution, ce qui pose un problème de passage à l'échelle des méthodes d'analyse. Deuxièmement l'interprétation à un niveau métier des évènements est difficile car elle requiert l'expérience et la connaissance métier de l'analyste, cependant ce niveau d'abstraction n'est pas explicite dans la trace. Nous proposons d'aider l'analyste en apportant des outils de gestion des connaissances pour naviguer dans la trace en utilisant des concepts métier de haut niveau.

La suite de cet article est organisée comme suit. Dans la section 2 nous proposons une ontologie du domaine de l'analyse de traces d'applications multimédia sur SoCs. Dans la section 3 nous présentons l'adaptation de l'ontologie à un cas d'utilisation réel. Dans la section 4 nous présentons l'analyse des traces du cas d'utilisation à l'aide de requêtes SPARQL. La section 5 conclut ce papier et présente nos travaux en cours.

BDA 14 octobre 2014, Grenoble-Autrans, France.

2. VIDECOM, UNE ONTOLOGIE POUR L'ANALYSE DE TRACES D'EXÉCUTION

L'analyste cherche à retrouver dans la trace les concepts métier correspondants à ses connaissances de l'application et de l'architecture du SoC. Cependant la relation entre les évènements de trace et ces concepts métier n'est pas explicite dans la trace. Nous présentons dans cette section notre approche pour enrichir les évènements de la trace. L'approche consiste à connecter les évènements de trace, à l'aide éventuellement des règles de déduction, à des classes et propriétés d'une ontologie représentant les concepts et contraintes pour l'analyse de traces issues de SoC.

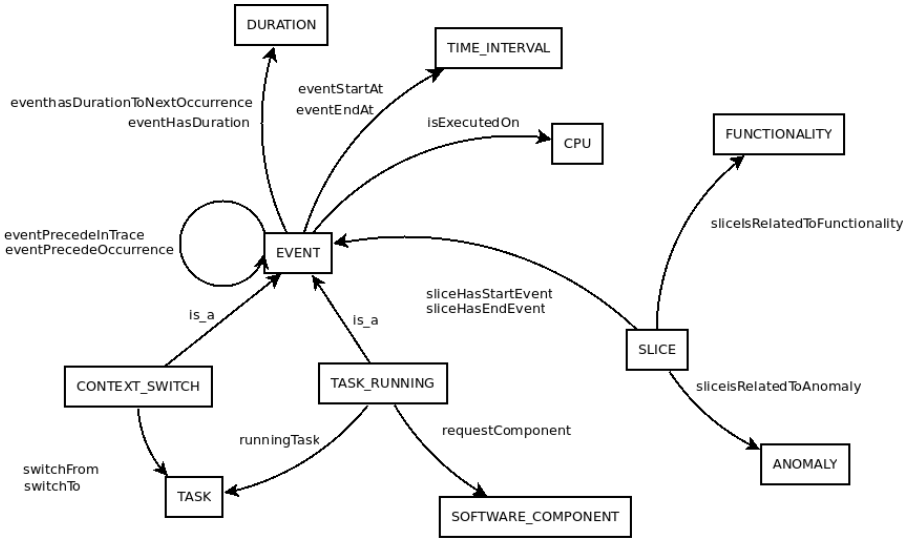

Figure 1: Extrait des classes et propriétés principales de Videcom.

2.1 Modèle de VIDECOM

VIDECOM est une ontologie légère, basée sur la sémantique de RDFS. Elle contient 608 classes et 238 propriétés représentant des concepts du domaine de l'analyse de trace d'application multimédia sur SoC.

La Figure 1 présente un extrait des classes et propriétés de VIDECOM. La classe principale Event représente les différents types d'évènements, on distingue TASK_RUNNING pour les évènements d'exécution de tâche et CONTEXT_SWITCH pour les évènements de changement de contexte entre deux tâches. Les classes FUNCTIONALITY et ANOMALY représentent respectivement les fonctionnalités et les anomalies de l'application. TIME_INTERVAL permet de représenter la temporalité des évènements et DURATION les différents types de durées. La classe SLICE sert à représenter une zone de trace. La propriété *isExecutedOn* capture la connaissance du processeur où se produit un évènement, *runningTask* représente la tâche exécutée par l'évènement et *requestComponent* permet de représenter les composants logiciels (interruptions, appels système ou appel fonction) sollicités par l'évènement. L'ordre entre les évènements est capturé par plusieurs propriétés : *eventPrecedeInTrace* indique l'évènement précèdent dans la trace tandis que *eventPrecedeOccurrence* indique l'occurrence précédente de l'évènement dans la trace. Ces classes et propriétés sont utilisées pour construire des règles métier qui permettent à l'analyste de décrire la sémantique des concepts et contraintes métier qu'il souhaite identifier dans la trace.

2.2 Implémentation efficace du triple Store

Pour exploiter VIDECOM dans l'analyse de trace, ses classes et propriétés sont instanciées à partir des évènements de la trace, puis saturées, stockées et requêtées dans un Triple Store [6]. Nous avons fait le choix d'une saturation à priori du Triple Store pour assurer des résultats complets aux requêtes. La saturation matérialise de nouvelles instances à partir des règles d'inférence RDFS et métier. A cause de la grande taille de la trace, le Triple Store résultant est de l'ordre de plusieurs dizaines de millions de triplets. Une étude comparative du passage à l'échelle de l'exploitation de ces triplets sur 7 systèmes de bases de connaissances (Jena, Sesame, Sesame-native, TDB, SDB, rdf-3x, vertical-mdb), a montré que la méthode du partitionnement vertical, introduit par Abadi [1], permet d'avoir des temps acceptable de chargement et de réponse aux requêtes sur 95 millions de triplets [7]. Cette méthode stocke les triplets dans des tables ayant le modèle relationnel suivant : $propertyP(subject, property, object)$. Chaque table représente une propriété de VIDECOM et contient tous les triplets correspondants à ses instances. Les tables sont physiquement stockées sous forme de collections de colonnes par des gestionnaires de bases de données orientés colonnes spécialement conçus pour le partitionnement vertical [2, 13].

2.3 Description d'un cas d'utilisation réel

Pour expérimenter notre approche, nous nous intéresserons au cas réel d'usage de l'analyse des traces d'une application d'enregistrement en streaming provenant de STMicroelectronics. L'application, nommée *ts_record*, exécute plusieurs fonctionnalités en parallèle. Tout d'abord, les données streaming sont collectées et stockées dans des buffers IP par une tâche t1, une seconde tâche t2 les copie ensuite des buffers IP vers la mémoire centrale. La dernière tâche t3 se charge enfin de les copier de la mémoire centrale vers le disque USB. Les tâches t1, t2 et t3 respectent des contraintes temporelles pour éviter de lire trop tard ou trop tôt des données. Par exemple la tâche t2 doit lire les données des buffers IP et les écrire en mémoire toutes les 100 millisecondes à l'aide de l'appel système *sys_write*. Le non respect de cette contrainte temporelle peut conduire à des anomalies dans l'application *ts_record* résultant en l'enregistrement d'un fichier corrompu sur le disque USB. Les traces de *ts_record* contiennent non seulement les évènements produits par l'exécution en parallèle des tâches t1, t2 et t3, mais aussi les évènements liés aux différentes activités du système d'exploitation.

IF	THEN
$(?e1, runningTask, ts\_record0)$ $(?e1, requestComponent, sys\_write0)$ $(?e1, eventPrecedeOccurrence, ?e2)$ $(?e1, eventHasDurationToNextOccurrence, ?period)$ $(?period == 100)$	$(\_ : s, sliceHasStartEvent, ?e1)$ $(\_ : s, sliceHasEndEvent, ?e2)$ $(\_ : s, sliceIsRelatedToFunctionality, sysWriteNormal);$

Table 1: Règle d'inférence d'une instance de zone de trace rattachée à la fonctionnalité sysWriteNormal

IF	THEN
$(?e1, runningTask, ts\_record0)$ $(?e1, requestComponent, sys\_write0)$ $(?e1, eventPrecedeOccurrence, ?e2)$ $(?e1, eventHasDurationToNextOccurrence, ?period)$ $(?period > 100)$	$(\_ : s, sliceHasStartEvent, ?e1)$ $(\_ : s, sliceHasEndEvent, ?e2)$ $(\_ : s, sliceIsRelatedToAnomaly, sysWriteBlocked);$

Table 2: Règle d'inférence d'une instance de zone de trace rattachée à l'anomalie sysWriteBlocked

2.4 Extension de VIDECOM au cas d'utilisation ts_record

L'analyste peut ajouter des sous-classes aux classes de base de VIDECOM pour l'enrichir avec les connaissances métier correspondant aux cas d'utilisation à analyser. Dans le cas de *ts_record* nous avons ajouté les sous-classes de FUNCTIONALITY suivantes : *dataFromEthernet2IP*, *dataFromIP2Memory, dataFromMemory2UB* et *sysWriteNormal*. Les trois premières correspondent aux fonctionnements respectifs des tâches t1, t2 et t3 de *ts_record* et la dernière correspond au respect de la contrainte temporelle. De même nous avons ajouté une sous-classe *sysWriteBlocked* à la classe ANOMALY qui correspond au non respect de la contrainte temporelle.

L'analyste utilise les règles d'inférence métier pour enrichir VIDECOM avec la sémantique des nouveaux concepts. Ces règles décrivent comment sont créées les instances de ces nouveaux concepts. Les règles d'inférence, que nous illustrons sous la forme conditionnelle IF_THEN, ajoutent les triplets de la clause THEN si les triplets ou les expressions de la clause IF sont présents ou vérifiés dans le Triple Store. La règle d'inférence du Tableau 1 illustre la règle d'inférence qui permet de rattacher une zone de trace à la fonctionnalité *sysWriteNormal* si elle correspond à une zone où la contrainte temporelle a été respectée. La règle du Tableau 2 quant à elle permet de rattacher une zone de trace à l'anomalie *sysWriteBlocked* si au contraire la zone ne respecte pas la contrainte temporelle. Les zones sont rattachées aux fonctionnalités et aux anomalies par les propriétés *sliceIsRelatedToFunctionality* et *sliceIsRelatedToAnomaly*.

3. EXPLOITATION DE VIDECOM

L'exploitation du Triple Store pour l'analyse de trace se fait à l'aide de requêtes SPARQL. Dans cette section nous allons l'illustrer dans l'analyser la trace de *ts_record*. La Figure 2 représente une interface pour directement exécuter des requêtes SPARQL sur le Triple Store.

3.1 L'exploration de la trace à l'aide de requêtes SPARQL

Nous allons commencer l'analyse de la trace *ts_record* en visualisant les différentes fonctionnalités ou anomalies de la trace. La requête R1 recherche toutes les zones de trace rattachées à des fonctionnalités ou à des anomalies dans la trace. Leurs temps de début et de fin respectifs sont également retournés pour les identifier dans la trace.

```
R1 :
SELECT ?slice, ?start, ?end, ?func, ?anomaly
WHERE {
    ?slice sliceHasStartEvent ?e1.
    ?slice sliceHasEndEvent ?e2.
    ?e1 eventHasStart ?start.
    ?e2 eventHasEnd ?end.
      OPTIONAL
{?slice sliceIsRelatedToFunctionality ?func.}
      OPTIONAL
{?slice sliceIsRelatedToAnomaly ?anomaly.}
}
```

La Figure 3 représente l'illustration des résultats de la requête R1. Cette illustration permet de rapidement identifier des zones intéressantes pour une analyse plus fine. La requête R2 sert à identifier les tâches exécutées sur le processeur cpu0 dans l'intervalle de 152537756 à 194882669 millisecondes, correspondant à la zone intéressante sélectionnée dans la Figure 3.

```
R2 :
SELECT ?task
WHERE {
  ?event eventStartAt ?start.
  ?event eventEndAt ?end.
  ?event runningTask ?task.
  ?event isExecutedOn cup0
FILTER (?start >= 152537756 AND ?end  < 19482669)}
```

R2 permet d'isoler de façon déclarative les tâches exécutées sur un processeur précis dans un intervalle de temps. La requête R3 est un exemple de requête complexe permettant d'analyser un problème identifié dans la trace. Elle recherche dans tous les cas où une anomalie est déclarée sur un processeur, toutes les tâches qui s'exécutent sur les autres pro-

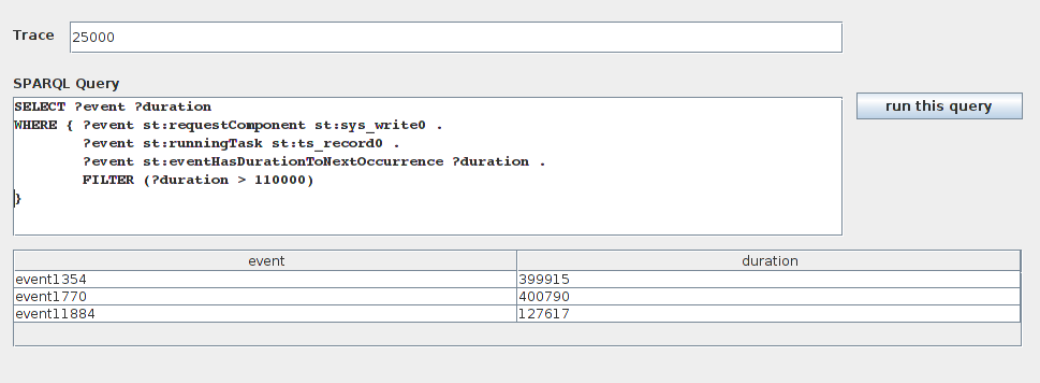

Figure 2: Interface pour exécuter des requêtes SPARQL sur le Triple Store.

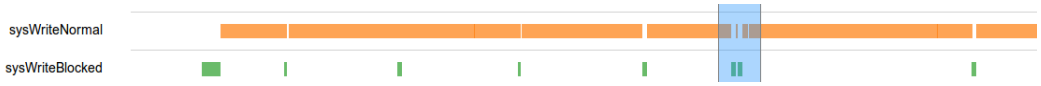

Figure 3: Time line représentant les fonctionnalités sysWriteNormal et les anomalies sysWriteBlocked.

cesseurs. Les résultats de cette requête ont permis de détecter que la tâche t3 s'exécute toujours en cas d'anomalie *sysWriteBlocked*. Une cause de l'anomalie s'est révélée être le fait que l'interruption levée par la tâche t3 pour lire les données en mémoire pouvait bloquer la tâche t2 si la même zone mémoire était sollicitée par les deux tâches. Ce qui occasionnait alors la perte des données des buffers qui n'étaient pas lu à temps.

```
R3 :
SELECT ?task ?cpu1
WHERE {
      ?slice sliceIsRelatedToAnomaly sysWriteBlocked.
      ?slice sliceHasStartEvent ?event1.
      ?slice sliceHasEndEvent ?event2.
      ?event1 eventStartAt   ?sstart.
      ?event2 eventEndAt   ?send.
      ?event1 isExecutedOn ?cpu0.
      ?event eventStartAt ?start.
      ?event eventEndAt   ?end.
      ?event runningTask  ?task.
      ?event isExecutedOn ?cpu1
      FILTER (?start >= ?sstart and ?end  <= ?send)
      FILTER (?cpu0 != ?cpu1)
}
```

3.2 Les performances

Nous avons utilisé MonetDB [12] comme gestionnaire de base de données orienté colonnes. Dans le but de rendre cette implémentation transparente à l'analyste, nous transformons toutes les requêtes SPARQL de l'analyste en requêtes SQL [5, 9] applicables au modèle relationnel du partitionnement vertical. De façon générale une conjonction dans la requête SPARQL est transformée en une jointure entre deux tables dans le partitionnement vertical. Nous avons effectué nos expériences sur une machine ayant un processeur de 2.27 GHz et 64 Go de RAM. Dans ces conditions nous avons traité une trace de 5 000 000 d'évènements correspondant à 25 minutes d'exécution de *ts_record*. Le Triple Store obtenu contient 95 309 610 triplets qui ont été saturés en 2 j 11 h 35 m 08 s à l'aide d'une implémentation SQL de l'opération de saturation. L'algorithme de saturation RETE implémenté dans Jena [10] est plus efficace mais nous n'avons pas pu l'utiliser car il nécessitait largement plus que les 64 Go de mémoire centrale. Les temps d'exécution des requêtes SPARQL varient entre 5 millisecondes pour les requêtes de type R2 où l'intervalle temporel est précisé et 3 minutes et 33 secondes pour des requêtes de type R3 qui s'exécutent sur toute la trace.

4. CONCLUSION

L'analyse des traces pour le débogage et l'optimisation des applications sur SoC est une tâche difficile, à cause de la taille importante des traces et du bas niveau des évènements. Nous avons proposé une approche pour enrichir la sémantique des évènements de traces en utilisant des concepts métier de plus haut niveau, représenté dans une ontologie du domaine de l'analyse de traces issues des SoC. Nous avons considéré un cas réel d'utilisation pour montrer l'intérêt de l'approche dans l'analyse de trace. Et nous avons montré qu'à l'aide de requêtes SPARQL, l'approche permet d'identifier rapidement des zones intéressantes et d'explorer plus finement la trace ce qui améliore l'analyse de la trace.

Bien que le temps de réponse aux requêtes reste abordable pour de grandes traces, la saturation à priori des triplets est une opération très coûteuse. Nous travaillons actuellement sur des techniques d'optimisation de la saturation. La première idée est de maintenir la saturation à priori dans ce cas nous avons expérimenté la distribution du Triple Store suivie de la saturation parallèle des portions. Cependant ces portions de Triple Store saturées n'assurent pas des réponses complètes aux requêtes à cause de la dépendance sémantique

des évènements. La deuxième idée est de réduire le nombre des évènements considérés dans la trace soit en effectuant un échantillonnage soit en considérant une première abstraction de ces évènements sous forme de patterns fréquents [11].

5. ACKNOWLEDGMENTS

Ce travail est financé par le projet FUI SocTrace. L'ontologie VIDECOM, des requêtes métiers ainsi que des triplets obtenus à partir de différentes de traces sont disponibles à l'adresse http ://videcom.imag.fr

6. REFERENCES

[1] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases*, pages 411–422. VLDB Endowment, 2007.

[2] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Sw-store : a vertically partitioned dbms for semantic web data management. *The VLDB Journal The International Journal on Very Large Data Bases*, 18(2) :385–406, 2009.

[3] T. Ball. The concept of dynamic analysis. In *Software Engineering ESEC/FSE '99*, pages 216–234. Springer, 1999.

[4] T. Ball and J. R. Larus. Optimally profiling and tracing programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(4) :1319–1360, 1994.

[5] R. Cyganiak. A relational algebra for sparql. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*, page 35, 2005.

[6] C. David, C. Olivier, and B. Guillaume. A survey of rdf storage approaches. *ARIMA Journal*, 15 :11–35, 2012.

[7] L. C. Fopa, J. Fabrice, A. Termier, T. Maurice, and I. Oleg. Benchmarking of triple stores scalability for mpsoc trace analysis. In *2nd International Workshop on Benchmarking RDF Systems*, 2014.

[8] A. Hamou-Lhadj and T. C. Lethbridge. A survey of trace exploration tools and techniques. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 42–55. IBM Press, 2004.

[9] S. Harris and N. Shadbolt. Sparql query processing with conventional relational database systems. In *Web Information Systems Engineering–WISE 2005 Workshops*, pages 235–244. Springer, 2005.

[10] A. Jena. A free and open source Java framework for building Semantic Web and Linked Data applications. `https://jena.apache.org/`, 2011. [Online ; accessed 10-Febuary-2015].

[11] C. Kamdem Kengne, L. C. Fopa, A. Termier, N. Ibrahim, M.-C. Rousset, T. Washio, and M. Santana. Efficiently rewriting large multimedia application execution traces with few event sequences. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1348–1356. ACM, 2013.

[12] MonetDB. Monetdb colums-store pioneers. `https://www.monetdb.org/`, 2008. [Online ; accessed 10-Febuary-2015].

[13] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al. C-store : a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.