Special issue：Information Platform

Research paper

Multimedia Device Information Engine for Symbiotic Applications

共生アプリケーションのためのマルチメディア・デバイス情報エンジン

Frédéric, ANDRES
National Institute of Informatics
フレデリック アンドレス
国立情報学研究所
Kinji, ONO
National Institute of Informatics
小野 欽司
国立情報学研究所
Hideaki, TAKEDA
National Institute of Informatics
武田 英明
国立情報学研究所

ABSTRACT

Multimedia device networks are being widely deployed for emerging applications mediating the symbiosis between human and informatics data. In these new applications, exploration and interaction are not only done by end-users alone, but it also actively implicate computerized environment often called Symbiotic Systems. Symbiotic Systems generally consist of large numbers devices (robots, surveillance devices, sensors…), each of which has diverse characteristics and interacts selectively in dynamic manner. It is generally high-speed computerized network sustaining these environments. Both long-running queries and short-running queries coexist. Symbiotic Applications rely on distributed system for collecting and exchanging device data. However, these systems are still emerging and they have a lack of customizability for data collection and management. They also don't scale to a large number of devices because large volumes of raw data are transferred without query-type optimization. In our approach of Multimedia Device Information Engine, queries decide which data should be extracted from the devices. In this paper, we define a model for Multimedia Device Information Engine. We also describe the design and the implementation of the BT Engine prototype.

要旨

マルチメディア･デバイス･ネットワークは人間と情報システムの共生を仲介する新しいアプリケーションにおいて広く使われるものである．これらの新しいアプリケーションにおいては，情報探索や情報との相互作用はエンドユーザ単独で行なわれるのではなく，シンビオティックシステムと呼ばれるコンピュータ化された環境を含めて行なわれるものである．シンビオティックシステムはロボットや監視装置，センサなど多数のデバイスからなり，そのそれぞれが独自の特徴をもち動的に相互作用を行なう．このような環境には高速のコンピュータネットワークが使われる．このネットワークにおいては長期にわたるクエリと短期で終わるクエリの両方がある．シンビオティックシステムにおけるアプリケーションは分散システム，すなわち各種のデバイスのデータを収集したり交換するシステムの上に実現される．しかし，このようなシステムはまだ新しいものであり，データ収集やマネージメントに対する柔軟な取り扱いが解決しなければならない課題である．本研究のマルチメディア･デバ

イス情報エンジンのアプローチでは，クエリ自身がデバイスからどのデータを取り出せばよいのかを決定することができる．本論文ではこのマルチメディア・デバイス情報エンジンのモデルを定義する．またこのシステムの設計と実装についても言及する．

[Keywords]

Device Information Engine, Multimedia, Fault Tolerance, PHASME, Open Platform

[キーワード]

装置情報エンジン，マルチメディア，フォールトレランス，PHASME，オープンプラットフォーム

1 Introduction

The widespread deployment of mobile devices is transforming Internet into InterSpace[1] area, “Symbiotic zones”, or cyberspace[2]. Symbiotic Devices such as robots, surveillance devices, sensors…do not only produce multimedia data (text, images, sound, video, …) but they also embed computing and communication capabilities. Furthermore, devices are able to store, to process locally and to transfer multimedia data they produce.

Symbiotic Applications monitor Interspace area by querying and analyzing multimedia data produced by devices. Examples of Symbiotic applications include remote robot supervision, information collecting in remote area. Typically, these applications involve information related to the device network and device multimedia data. Such information and data are stored in what we call Multimedia Device Information Engine.

This paper focuses on device query processing – the design, algorithms, and implementations used to run querying over multimedia device information engine as part of symbiotic systems. Any query issued by a device is called in the following of the paper, device query. The concepts developed in this paper have been developed under the COE Symbiotic project[3]. We define a device query as a query expressed over a device information engine.

As in relational databases, queries are easiest to express at the logical level. Queries are formulated regardless of the physical structure or the organization of the device network. The actual structure and population of the device network (e.g. number of connected robots or symbiotic devices) may vary over the lifespan of any query.

Clearly, there are similarities with relational database query processing. Most symbiotic applications combine device data with stored data. Device data refer to data created by the device itself. Stored data come from outside of the device. However, the features of device queries described here do not lend themselves to easy mapping to relational databases and device data is different from traditional relational data (since it is not stored in a database server and device data varies over the time).

There are two approaches for processing device queries: the warehousing approach and the distributed approach. The warehousing approach represents the current state-of-the art. In this approach, processing of device queries and access to the device network are separated. The device network is simply used as a data collection mechanism. The warehousing approach proceeds in two steps. First, data is extracted from the device network in a predefined way and is stored in information engine located on a unique front-end server. Subsequently, query processing takes place on the centralized information engine. The warehousing approach is well suited for answering predefined queries over historical data.

In the distributed approach, the query workload determines the device data that should be exchanged between the devices. The distributed approach is thus flexible – different queries can extract different multimedia data from the device network. In addition, the distributed approach allows the device information engine to leverage the computing resources on the device nodes: a device query can be evaluated at the front-end server, or in the device network, or at each device, or at some combination of the three.

In this paper, we describe a device information engine in a distributed environment and the present the choices we have made in the current implementation of our Device Information Engine called the BT engine. This paper makes the following contributions:

1. We propose a data model and long-running query semantics for any device information engine. As we previously mentioned, a device information engine mixes stored data and device data. Each long-running query defines a persistent view, which is maintained during a given time interval.

2. We describe the design of our BT Device Information Engine. The BT Engine extends the PHASME application-oriented system. Each device can customize the BT engine according to plug-ins. Queries can be formulated either in SQL or in PHASME Interface Language. We extended the query execution layer with new mechanism for the execution of device plug-ins to support long-running queries.

Addressing these issues is the necessary first step towards a device information engine. In addition, a device information engine should take into account devices and communication failures; it should consider device data. We believe that these challenging issues can only be addressed once the data model and international representation issues have been involved. In Section 2, we propose a model for Device Information Engine. Section 3 introduces the BT engine and section 4 overviews the resource management of BT. Then the approach on the Fault Tolerance management is described in Section 5. Section 6 introduces an overview of previous works. Finally, Section 7 concludes.

2 A Model for Device Information Engine

In this section, we will define more the features related to the notation of “device databases” and device queries. The PHASME system[4] is extended to support a data model for device data and algebra of operators to formulate device queries.

2.1 Device Database

A device database includes stored data and device data as it has been mentioned. In addition, stored data include the set of devices that participate in the device database together with characteristics of the devices (e.g. device’s location) or characteristics of the Interspace environment. The stored data are best represented as binary relations. The key issues are: How to represent device data? First device data are generated by device processing functions. Second, representation for device data should facilitate the formulation of device queries (data collection, time correlation, and aggregates over time windows).

Time plays an important role. Possibly, device-processing functions return output repeated over the time, and each output having a time-stamp. In addition, querying optimization introduces constraints on the device data time-stamps.

Given these constraints, we represent the device multimedia data as time series. The representation of the device time series is based on the Extended Binary Graph (so called EBG) structure[5]. The EBG structure provides the storage for the device data that includes ordering. EBG operators are n-ary mappings on the EBG structure. All EBG operators can be composed. In addition, they include the following functions: select, find, search, join, and aggregates over a set of positions.

We represent device data as time series with the following properties:

1. The set of records corresponds to the outputs of device function over the time.
2. The ordering domain is a discrete time scale.
3. All the outputs of device-processing functions are associated to a position.
4. Whenever a device function produces an output, the base sequence is updated at the physical position corresponding to the production time.

2.2 Device queries

Device queries involved stored data and device data, i.e. binary relations and time-based data. We define a device query as an acyclic graph of EBG operators. The inputs of an EBG operator are EBGs or the results of another EBG operator.

A device query defines a persistent EBG during its associated time interval. So, this persistent EBG is maintained to reflect the updates on the multimedia device information engine. Any EBG can be maintained incrementally without accessing to the complete content. This property is hold in our device information engine.

3 The BT Device Information Engine

In this section, we discuss the representation of device data, as well as the way of device query management. Furthermore, we discuss the limitations of the target BT system (see Figure 1) and how we will go over them.

We have introduced in Section 2 a model of device information engine. Two major elements influenced our research:

1. User representation: How devices and device processing functions can be modeled in the data schema? How can we formulate device queries?
2. Internal representation: How is device data represented within the information engine components from the definition layer to the query execution layer? How can we classify device queries in short and long-running query type?

3.1 User representation

In the BT multimedia device system, processing functions are represented as plug-ins following application-oriented implementation[5]. This approach improves traditional Object-relational databases ADT (so called Abstract Data Type) support as it has been pointed out in[6]. A device plug-ins is defined for compatible devices. The public interface of a device plug-ins includes specific data processing supported by one type of device. One plug-ins in the information engine corresponds to one device. Device queries can be formulated in SQL in order to be compatible.

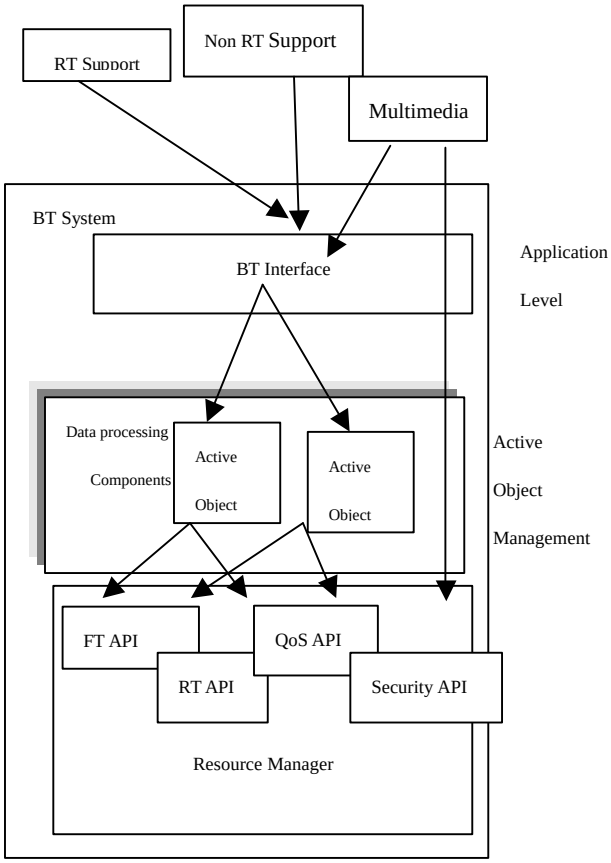

Figure 1: The BT System

3.2 Internal Representation

Query processing takes place on one multimedia information engine front-end while data processing functions are executed on the device nodes involved in the query. Figure 2 points out the resource manager of the BT system. The resource manager includes mechanism (resource planer, admission controller, resource allocation manager) for interacting with remote devices and resources. On each device, a lightweight information engine is responsible for execution processing functions and sending back data to the front-end.

In BT, we assume that there are no modifications to stored data during the execution of queries. A transaction manager based on strict two-phase locking is implemented on the information engine front-end. Each query can be classified as short or long-running query. Queries are considered as persistent views.

3.2.1 Shortcomings with traditional ADT function execution

In current object-relational database systems, ADT functions are used to form expressions combined with constants and variables. When an expression including an ADT function is evaluated, a function is called to obtain the return value. It is mostly assumed that this return value is available on-demand. This assumption does not hold in a device information engine for the following reasons:

1. Device ADT functions incur high latency due to their location or because they are asynchronous;
2. When evaluating long-running queries, Device ADT functions return multiple outputs.

3.2.2 Virtual EBGs

To overcome the problems previously outlined, we introduced an EBG operator to model the execution of device ADT functions. This operator is a variant of the join between an EBG that includes the device item and the device plug-ins. The representation of the device plug-ins is called a virtual EBG.

A virtual EBG is a plug-ins. It follows the definition of plug-ins of application-oriented information engine.

The observation of symbiotic applications gave the following inputs:

- A virtual EBG is append-only: new data are appended to the virtual EBG when the associated device processing function returns a result. Data in virtual EBGs are never updated or deleted.
- A virtual EBG is naturally partitioned across all devices represented by the same device plug-ins. A virtual EBG is associated to a device plug-ins, to each device of these type is associated a fragment of the virtual EBG.

A device information engine is internally represented as a distributed database. Virtual EBGs are partitioned across a set of devices. Based EBGs are stored on the information engine front-end. Distributed query processing techniques are based on the application-oriented concept.

3.2.3 Query Execution Plan

Virtual EBGs appear in the query execution plan at the same level as base EBGs. Base EBGs are accessed through EBG operators. Each fragment of Virtual EBGs is accessed on devices using virtual operators. A virtual operator incorporates in the query execution plan the adequate mechanism necessary to support long-running queries.

3.2.4 Internal Uniform Task Model

The resource management follows a task model that yields a uniform presentation of applications within the BT system. Such model needs to be flexible enough to express the stringent timeliness requirements of different applications. Furthermore, it must cope with the complexity of multimedia tasks, such as variable bit rate compression of video streams. The major feature is a single abstraction for reasoning about multimedia and real-time system.

Let define A $[t, t+\tau] = \int_{t,t+\tau} a(x)dx$

as the execution time necessary to complete the workload that arrives for the system in the time interval [t, t+ τ].

Then the workload that arrives for a task can be characterized by a task envelope A* which provides an upper bound on A, that is , for all times $\tau \geq 0$ and $t \geq 0$ as it has been shown in [7]:

(1) $A[t, t+\tau] \leq A^*(\tau)$

A task envelope A* should be sub-additive, that is, should satisfy

(2) $A^*(t_1) + A^*(t_2) \geq A^*(t_1+t_2)$

$\forall t_1, t_2 \geq 0$

if a task envelope A^*_1satisfies (1) but is not sub-additive, it can be replaced by the sub-additive task envelope A^*_2 such that $A^*_2(t) \leq A^*_1(t)$ for all $t \geq 0$.

The processing requirements of a task are described as follows: Let $S[t,t+\tau]$ denote the amount of service time that a processor can devote to processing one or more instances of the task in time interval $[t,t+\tau]$. With the task envelope A* and a deadline for completing an instance of this task, say D, a task always meets its deadline if for all $\tau \geq D$ we have: $\min \{A^*(t-\tau) = S[0, t]\} \leq D$. We use S*(t) = A*(t - D) to denote the service envelope of the task. If a processor can guarantee for each time interval of length t that a task obtains a service of at least S*, then a deadline violation will never occur.

4 The Resource Management

Critical component of the BT platform is its ability to manage a large, globally distributed set of resources efficiently. Therefore, the BT Platform requires end-to-end resource management, including physical resources (such as devices), end systems resources (such as CPUs, memory, and network interface virtual circuits), and communication resources (such as link bandwidth). We assume that the network and operating systems provide resource management.

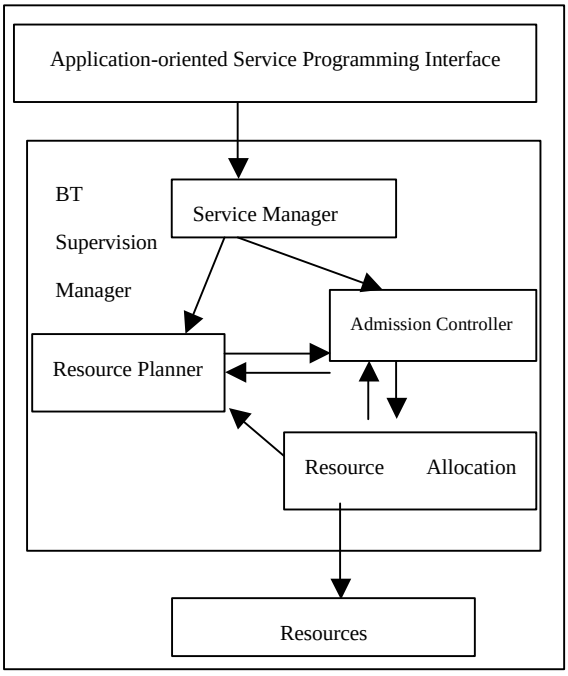

Figure 2: The Resource Manager of the BT System

Based on this primitive resource management, the Harmony platform will provide an end-to-end adaptive resource manager (see Figure 2) that supports applications with widely varying QoS requirements, such as requirements on timeless, fault tolerance, and security. The following support is needed inside Harmony's resource manager to achieve these capabilities:

Support for an application-oriented service-programming interface that enables application programmers to specify their desired QoS without requiring knowledge of the underlying low-level resource management entities.

Support for a mapping mechanism to transform qualitative and quantitative application-oriented service requirements into quantitative resource allocations. The service manager generates execution plans of the service-dependent QoS requests from applications according to the uniform task model described in the following of this section.

Support for a dynamic network and system resource management that can maximize global system resource utilization.

Support for admission control to check if the resource and QoS requirements of the new application are acceptable for The Harmony System without compromising the QoS guarantees made to currently applications.

Maximizing global system resources is related to the use of a planning component. The planning component keeps track of the dynamic behavior of resource usage. By maintaining information not only of the current state of resource utilization, but also of past and (predicted) future usage, the resource management scheme can adapt to the changing resource demands in the entire system. This feature is particularly important during peak loads, when the responsiveness of the system is particularly essential to ensure mission- critical response.

Application-oriented Service Interface

The application-oriented Service interface (AOS API) provided by the Harmony platform is designed to satisfy several constraints. First, this API allows application developers to access the full functionality of the Harmony platform without being burdened with internal system details, such as scheduling or primary/secondary memory management. Second, the API must be simple enough to provide an intuitive and customizable internal representation. Thus, the trade-off of such an API is between the requirements for application customizability and internal simplicity.

5 Adaptive Fault Tolerance Management

Given the large and growing size of device networks, faults may occur frequently and at inopportune times. Adaptive fault tolerance management, such as the approach being defined for the OMG specification is needed to protect the system against common security breach points.

A first approach can follow an adaptive information engine solution for non-malicious faults. The following fault hypotheses will be considered: processors may fail silently (multiple failures are possible); transient faults may occur due to power glitches, software bugs, and timing faults can occur where data is out of date or not available in time. If the global virtual information engine can handle these faults and operate efficiently, then it should be robust and useful under some scenario.

Our solution proposes a service-oriented and adaptive fault tolerance management.

Service-oriented Fault Tolerance: the BT system will have a behavior according to the following services:

Read only queries: Such kind of workload can be either dynamically requested by device or automatically triggered by the actions in the active data processing part of the BT System. These queries can have soft deadlines and can retrieve any kind of data types (text, audio, video).

Update transactions: These transactions can be user invoked or automatic. They update any type of data including temporal data according to the permission.

- Write transactions.
- Multimedia devices: Such devices have time constraints, are large in volume, include synchronized, and can be degraded in term of resolution if necessary.
- Analysis devices: Such device received data for processing requirements in distributed environments.

Support for Adaptive Fault Tolerance: device information engine may access any number of objects. To support fault tolerance services, the underlying system model needs to be based adaptive (secure) fault tolerant (real-time) objects. The fault tolerance can become very expensive, the cost of fault

tolerance should be tailored according to the requirements of each device. In this approach, each object represents data and methods on the data and various types of semantic information that support adaptive (secure) fault tolerance in real-time.

Processing description: in addition to the parameters required for its functionality, the input of an object can be the time requirement, the QoS requirement, the degree of fault tolerance, and the level of security. Control modules are inside the objects and transparent for the devices. They are used to meet the incoming requirements dynamically based on the request and the current state of the environment.

Again, also for this service, one key research issue is the mapping of the service level of the fault tolerance request to the underlying objects. Given the underlying object mechanisms that support adaptive fault tolerance, the key issue is the objects composition to meet the service level requirements.

6 Previous works

Several works have done in the field of symbiotic systems such as[7-9]. They try to take more synthetic approach to include basic principles of intelligence in the context of information systems. In addition, they propose new underlying principles of intelligence using high degree by using symbiotic systems with multiple perception channels, such as vision, auditory, tactile, etc.

7 Conclusions

Device information engines are a promising new field for database research. We described a data model and long-running query semantics for device database systems where stored data are represented as relations and device data are presented as sequences. The BT system that we presented is the 1st effort toward a device information engine. This approach demonstrated that the application of database technology shows much promise for providing customizable and scalable access of large collections of devices as part of symbiotic systems. It enabled us to identify a set of challenging issues that we are addressing with our ongoing research in the Symbiotic COE project:

- Due to large scale of a network of device, it is important to include failure tolerance as some of the devices and some of the communication links will fail at some points during the processing of long-running queries. One issue is the way that device information engines can be optimized.
- Device data can be classified as measurements. We are defining a data model and related operators for representing and manipulating continuous distributions.
- Because of the large scale and dynamic nature of a device network, metadata management requires a layer of metadata servers. This study concerns the way to manage distributed metadata and the way to utilize the information to devise good query plans.

Acknowledgements

We would like to thank Professor Haruki Ueno who helped to start the initial research work of this paper. Thank you also to all the reviewers for their valuable comments.

References

[1] http://www.canis.uiuc.edu/projects/interspace/

[2] Anderson, C., “Cyberspace offers chance to do 'virtually' real science”, *Science*, vol. 264, pp. 900-901, May, 1994.

[3] Ampornaramveth, Vuthichai; Ueno, Haruki, “Introduction to Research on Symbiotic Information Systems and an Agent-Based Robot Manipulator Sharing over the Internet”, *Proc. WAINS 7*, pp. 145-154, 2000.

[4] Andres, F.; Ono, K., “The Distributed Management Mechanism of the Active Hypermedia Delivery System platform”, *IEICE Transactions*, August, 2001.

[5] Andres, F.; Ono, K., “Phasme: A High Performance Parallel Application-oriented DBMS”, *Informatica Journal*, Special Issue on Parallel and Distributed Database Systems, Vol.22, pp. 167-177, May, 1998.

[6] Andres, F.; Boulos, J.; Ono K., “Accessing Active Application-oriented DBMS from the World Wide Web”, *Proceedings of the Intern. Symp. on COoperative DAtabase Systems for Advanced Applications (CODAS)*, pp. 232-234, Dec. 1996.

[7] Kinoshida, T.; Sugawara, K., “ADIPS Framework for flexible Distributed Systems”, *Proc. Pacific Rim International Workshop on Multi-Agents (PRIMA '98)*, pp. 161-175, 1998.

[8] Imielinski, T.; Goe, S. l., “Data space -querying and monitoring deeply networked collections of physical objects”, *Proc. of the Int. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'99)*, Seattle, WA, August 1999.

[9] Hoff, J.; Bekey, G., “An Architecture for Behavior Coordination Learning”, *IEEE International Conference on Neural Networks*, Perth, Australia, vol. 5, pp. 2375-2380, November 1995.