Position Paper: Towards A New High-Level Architecture For ERP Systems

Christian Stefansen Jakob Grue Simonsen
Ken Friis Larsen
Department of Computer Science
University of Copenhagen
{cstef|simonsen|kflarsen}@diku.dk

August 29, 2007

Abstract

In this article we describe our thoughts for the core of an architecture for next generation ERP systems. We describe which properties we believe are important for future ERP systems, and argue how our architecture will provide those properties.

1 Introduction

The past few years have seen a shift toward *service-oriented architecture* and proces-orientation. Due to the fact that processes have not traditionally been represented directly in the ERP system, there has been a challenge for ERP systems providers to accomodate. This problem has been exacerbated by the heterogeneity of ERP systems. ERP systems were originally based on a general ledger, but over time several modules have been added, each with different ways of handling and representing business data. This has made querying for business intelligence prohibitively time-consuming. More specifically, current ERP systems suffer from a number of drawbacks:

- mismatch/tension between ***transactional data and destructive CRUD (create/read/update/delete)-accessed data
- interaction protocols are neither explicit nor exposed to the environment
- prevalent relational database design (including normalization) does not lend itself to business intelligence that straddles several functional silos.
- customizations are programmed in system-specific custom made organically evolved programming languages, which in turn means that upgrading the system is a manual, error prone, expensive, and time-consuming process.

To alleviate these problems we present a speculative high-level architecture with the following distinguishing features:

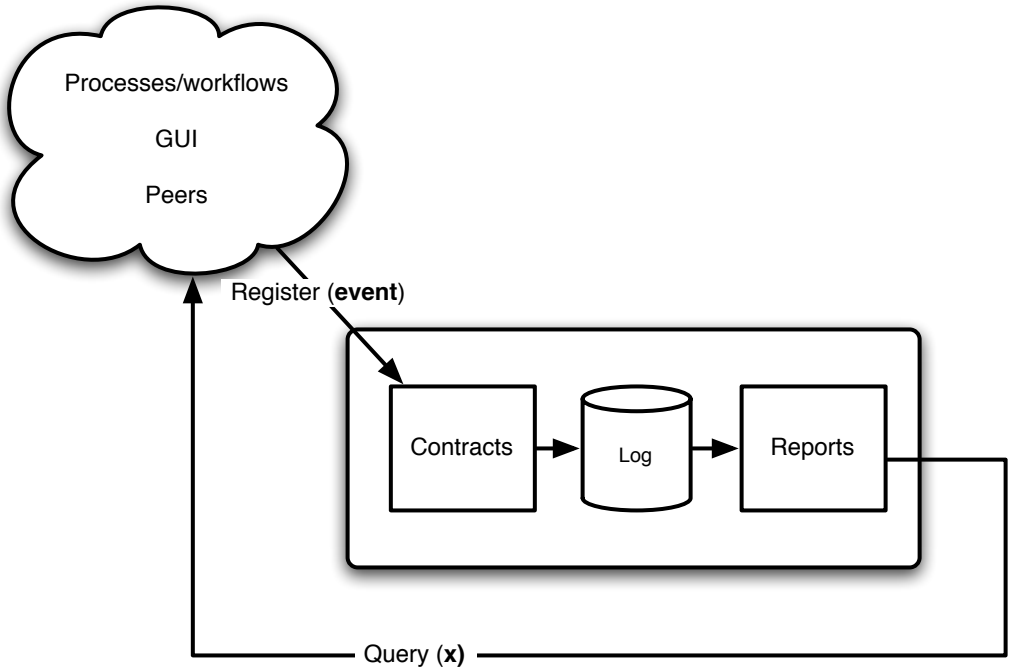
- *Flight recorder* property. That is, the ability to revert to any point in time and view the reports as they were at that time. In essence this takes the important idea from double-entry bookkeeping that one can only add entries—never delete—and makes it as a fundamental principle of the entire system.
- Interaction specifications are formalized and exposed to the environment as contracts

- Reports, template basis, not constrained by a database design that is static or cumbersome to change, minimal ontological bias. With this proposal the database structure becomes implicit in reports rather than being explicit (and static!) in the database.
- Adaptation to the user's needs are mainly done through *configuration* rather than customization in the ERP system's source code.

2 Overview

The high-level architecture is shown in Figure 1. It is based on the premise that the core of any ERP system—regardless of its particular domain of use—can be understood as a recorder of *events* as well as the *reports*, which—by processing and aggregating the event history—allow the environment to observe the state of the system.

Figure 1 High-level architecture



We partition our architecture into what we consider the essential parts of an EPR system:

- *The Core.* The event recorder. This part contains the contract descriptions, the event log, and the report repository.

  The core is what handles the core part of a business. Namely, recording all events so that the flight recorder property. But not only the events should be recorded but also the state of all contracts, so that it is possible to go back to any point in the past and what rights and obligations the enterprise had at that time. The reports are in the core to enable that the event log is optimised automatically and so that efficient summary data-structures also can be derived automatically.

- *User Interaction.* The ERP system should keep track of who is allowed to do what and what informations they are allowed to see, also know as *role-base UI*.

- *Workflow Management.* In what sequence should the execution of the enterprise's events take place.

The last two parts, while essential for a real ERP system, are outside the scope of this article.

2.1 Example of Event Sequence

Suppose that the following event occurs:

1. The company ships goods and issues invoice 123 of $100.

First, the event is sought matched to one of the contracts. If we have an ongoing sales process contract, the event can be matched to this contract, i.e. we register that the event has satisfied one step in the contract. Since a matching was successfully made, the event is committed to the log of events, and the state of the contract is changed to reflect that the event has occurred.

Now assume that another two events occur:

1. The company ships goods and issues invoice 124 of $50.
2. Payment of $50 for invoice 124 is received.

Assuming that the events can be matched to appropriate contracts, the event log now contains three events. Notice that the ERP system only registers what indisputably has taken place. So far the events have not been interpreted or made subject to a particular accounting principle, domain model, etc.

Suppose users of the ERP system wish to know the sum of outstanding debt. For this the system needs a report that goes over the log and sums up all events pertaining to invoicing and receipt of payments from accounts receivable. In our current example, it would thus compute a total debt of $100. This information is in turn used by users who after a while could generate, say, a payment reminder event.

3 Event handling

Currently we classify events into three classes:

(a) *All* events stipulated in a (real world) contract must be registered.

(b) Events in connection to a contract, that is not mentioned in the contract. But for which it can later be of interest to observe the event. That is, make a query for the information carried by the event.

(c) Event not specific to a contract, specific:

- Entering a new contract (which must then be added to the contract repository)
- Updating of master data
- Updating of a reporting function

4 Conclusion and future work

To support the claims put forth in this position paper, the following needs to be done:

1. A formal model for contracts should be developed and finalised (see [1] for an example of a contract specification language.)
2. A report language should be developed and finalised.
3. The architecture needs substantial work to scale and perform well.
4. Integration with the other essential parts of an ERP system should be investigated.

5 Acknowledgement

Most of the ideas presented in this article have been formed in cooperation with Fritz Henglein.

References

[1] Jesper Andersen, Ebbe Elsborg, Fritz Henglein, Jakob Grue Simonsen, and Christian Stefansen. Compositional specification of commercial contracts. In *1st International Symposium on Leveraging Applications of Formal Methods*, 2004.