Reactive Rules Inference from Dynamic Dependency Models

Asaf Adi, Opher Etzion, Dagan Gilat, Royi Ronen, Guy Sharon and Inna Skarbovsky
IBM, Haifa Research Laboratory
{adi, opher, dagang, royi, guysh, inna}@il.ibm.com

ABSTRACT

Defining dependency models is sometimes an easier, more intuitive way for ontology representation than defining reactive rules directly, as it provides a higher level of abstraction. We will shortly introduce the ADI (Active Dependency Integration) model capabilities, emphasizing new developments:

1. Support of automatic dependencies instantiation from an abstract definition that expresses a general dependency in the ontology, namely a "template".
2. Inference of rules for dynamic dependency models, where dependencies and entities may be inserted, deleted and updated.

We use the eTrade example in order to exemplify those capabilities.

Categories & Subject Descriptors:

Primary C.4 [**Computer systems organization**]: Performance of Systems – *modeling techniques, measurement techniques.*

Secondary C.3 [**Computer systems organization**]: Special-Purpose and Application-Based Systems – *signal processing systems.*

General Terms: Algorithms, Management, Design, Theory.

Keywords: dependency models, reactive rules, active systems, active databases, rule engine, event correlation, relationships between entities.

1. INTRODUCTION

Dependencies of many types are common in enterprise systems. They express the exact way in which entities affect other entities, and closely related issues were pointed in [1] as a major challenge. Some examples are:

1. An internet access provider allocates bandwidth for clients considering the current load and the type of service level agreement that each customer has. Any change in those requires re-calculating the allocation. This is a value dependency: it defines how the allocation value depends on other values.

2. In order to properly function, an internet site must have both its WAS and its DB server working.

This is a business-logic dependency: It defines a constraint that has to be met in order to ensure the availability of the site.

WWW 2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

Currently, there is no tool that can provide a single seamless view of all dependencies in an enterprise. Dependencies are dealt with using ad-hoc tools that are usually not integrated or synchronized with the rest of the enterprise system. It is thus impossible to predict what the consequences of some change will be, or to have efficient root-cause analysis. The vision is having an easy to use tool for modeling multi-level dependencies, intelligently infer reactive rules from the model and visualize it. ADI provides a higher abstraction level than reactive rules, concentrating on dependencies.

2. ADI MODEL AND DEVELOPMENTS

ADI models an ontology using entities and dependencies between them. A comprehensive description of ADI can be found in [2]. In general, ontology modeling consists of:

1. Defining the types of the entities and dependencies that can exist in it ("Dependency Types" and "Entity Types" in [2]).

2. Defining instances of those dependencies between instances of entities in a way that reflects the ontology. This is the actual Model ("Facts" in [2]). A new development is the support of general abstract dependencies (see section 2.2.).

3. Defining the effects that input events have on the system. An effect can update data in an entity, create/delete an entity or trigger an event ("Effects" in [2]). A new development is the support in insertion and removal of entities and dependencies from the model during runtime, and not solely on startup (see section 2.2.).

We discuss those developments using the eTrade example.

2.1 eTrade example

A website provides online stock trading services. The trading process is composed of: Transaction tasks (buy, sell), View Portfolio, Login and Logout. The first two are operated by a trading application, while the latter two - by an authentication application. This is modeled by a *mandatory dependency* between the services and the related applications as illustrated in figure1. Login and Transaction tasks are crucial for the proper functioning of the site. Therefore it depends on them in a *mandatory dependency*, which means that the site fails to function if at least one of them fails. Logout and View Portfolio are also important, but the site can function even if one of them fails. Therefore, the modeling requires *one-out-of dependency*, whose positive result is *mandatory* for the site. The trading and authentication applications depend, in a *mandatory* manner, on a DB server and

on *two-out-of* three WASes. Other examples for dependencies in ADI are arithmetic or referring to some percent of the sources.

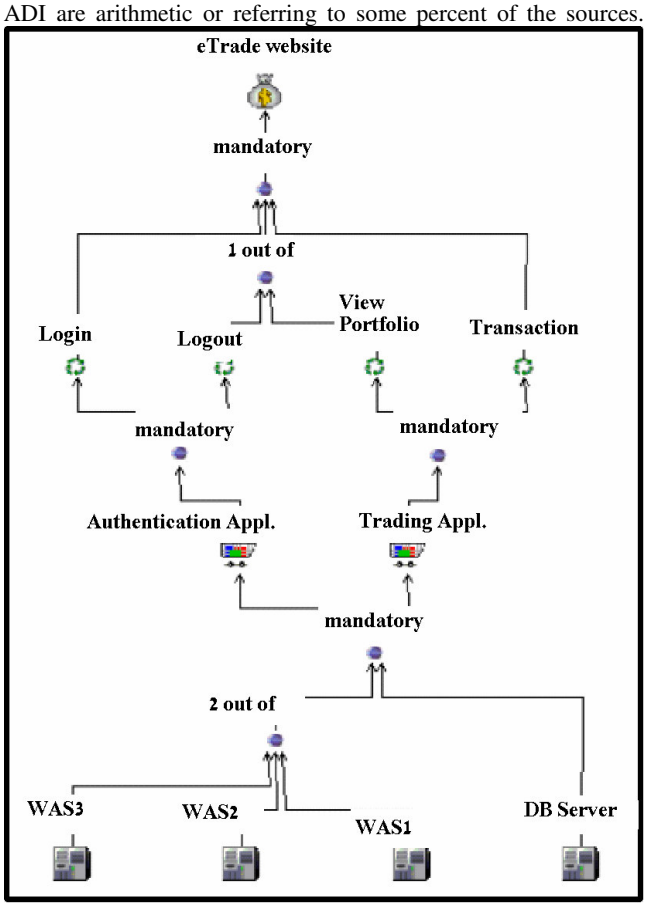

Figure 1 - The eTrade Dependency Graph

2.2 Discussion

Suppose that the site decides to add another WAS. ADI supports the insertion and removal of entities or dependencies (by Effects) without having to change the model and deploy it again, but dynamically react to such changes. Since many monitoring systems have to deal with constantly changing topologies, not having to "stop the world", lose the context and deploy the new topology is an extremely useful feature. Changes in topology influence data as well, and information has to be propagated through the model.

This feature is at the heart of the other development. Suppose that our site has mirror sites in several countries. We would like to provide a mechanism that exempts the user from defining similar dependencies when they are obvious. The solution is defining an abstract dependency (a template) between entities. The actual dependency will be instantiated immediately when entities of that type are created, provided that they meet the conditions specified in the template. For example, our site will be able to define a rule stating that each couple of DB server and Authentication server that have the same ID (they belong to the same mirror site) will have a mandatory dependency between them. That dependency will be automatically instantiated during runtime, without losing context or having to deploy the model. A self-instantiating abstract dependency is in effect a general rule in the ontology. Having it exempts the user from defining an obvious dependency and giving it as input. In large systems or in cases where the model changes frequently, manual input of dependencies is impractical and error-prone.

The importance of the dynamic model with abstract dependencies is not only user convenience, build-time efficiency or support in very large models. It also adds to the domain of problems solvable by ADI problems whose topology is not bounded or not exactly known in build time (or both).

The execution of a model is performed as follows: On an event, all the effects defined for it are retrieved and executed. As a result, new entities might be added, existing ones removed or their attributes values changed. In any case, the dependencies these entities participate in or had participated in are notified on the change and have to be resolved. The result of this may affect depending entities and so the process resumes until all changes and impacts have been properly propagated through the dependencies. Newly created and modified entities are checked against all abstract dependencies they may participate in for the dependency instance with the same context. If such an instance exists, the entity is added to that instance as defined by the abstract dependency. If a dependency with the same context does not exist a new one is created and all the entities that have the same context are added to it.

Dependencies in ADI are resolved using AMIT [2,3], a reactive rule engine. Intuitively, one expects that a different set of reactive rules will be arranged for every model according to its topology. However, we have been able to develop a generic set of AMIT rules that always stays the same, but can handle any ADI model. The initial model and runtime changes are translated into events. The rules are built in a way that they can represent and resolve the dependencies in any model based on these input events, therefore using AMIT in a compact elegant manner.

3. REFERENCES

[1] Avi Silberschatz, Michael Stonebraker, and Jeffrey D. Ullman. Database Research: Achievements an Opportunities into the 21st Century. SIGMOD Record 25(1) 1996: 52-63.

[2] A. Adi, O. Etzion, D. Gilat, and G. Sharon, “Inference of Reactive Rules from Dependency Models”, *LNCS*, Springer-Verlag, Heidelberg, November 2003, Vol. 2876, pp. 49-64.

[3] A.Adi, O. Etzion: The situation manger rule language. RuleML-2002.