USING DATA MODEL PATTERNS FOR RAPID APPLICATIONS DEVELOPMENT

David C. Hay
Essential Strategies, Inc.

WHAT ARE DATA MODEL PATTERNS?

The book *Data Model Patterns: Conventions of Thought*[1] presents a set of standard data models describing standard business situations. These model "patterns" can be used in a variety of businesses in a large number of industries, for the simple reason that all businesses are in fact structured in very similar ways. The personnel function, accounting, and order processing are similar, regardless of who is doing them. For this reason, it is possible to develop a generic pattern and then apply it to many companies.

For example, Figure 1 shows a model of an ORDER. Any sales or purchase order for any company must look like this. Note that instead of entities for "customer" or "vendor", you have here PARTY. A PARTY is simply any person or organization that is of interest to the enterprise. Here each is playing the role of customer or vendor. Specifically, "each PARTY may be *a buyer in* one or more ORDERS," and "each PARTY may be *a seller in* one or more ORDERS." If the seller is one of our own departments, the order is (to us) a SALES ORDER. If the buyer is one of our own departments, it is (also to us) a PURCHASE ORDER. For clarity, PURCHASE ORDER and SALES ORDER are shown as sub-types, although this is not strictly accurate. Note that it is possible for the same PARTY (ourselves or someone else) to be *both* the *buyer in* and the *seller in* the same ORDER. Is that a SALES ORDER or a PURCHASE ORDER?

The model also shows that each ORDER must be *composed of* one or more LINE ITEMS, where each LINE ITEM must be *for the purchase of* one and only one COMMODITY TYPE.

WHAT IS RAPID APPLICATIONS DEVELOPMENT?

Rapid Applications Development (known these days as "RAD") is a wish. The applications development process is composed of so many steps that we want to believe that some are unnecessary. If we could limit ourselves to doing just the necessary ones, perhaps we could significantly cut the amount of time to develop systems.

[1] David C. Hay, *Data Model Patterns: Conventions of Thought*, Dorset House Publishers, New York:1996.

To be sure, many tools have been developed recently to speed up the process. In addition to CASE tools for strategic planning and analysis, there are tools which allow program code to be produced so quickly that if the result isn't quite what was desired, it can be redone equally quickly.

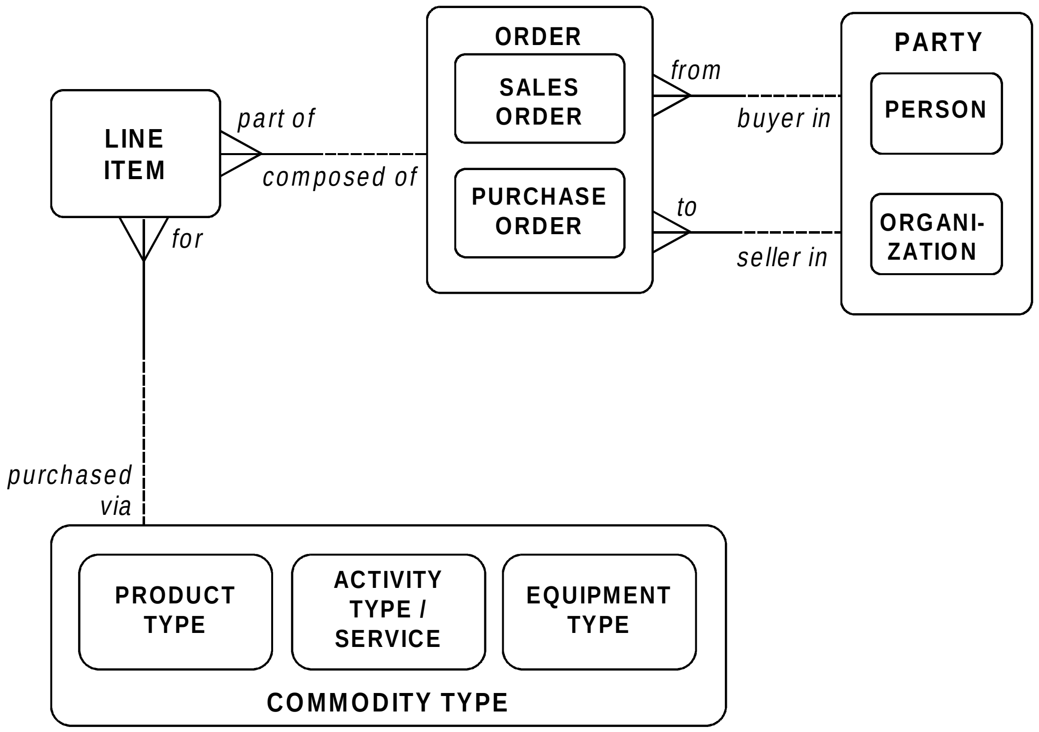

Figure 1: A Data Model Pattern

This latter development has encouraged developers to think that perhaps the time spent planning and designing systems could be short-circuited and all we would have to do is a series of pilot developments and corrections. A system would be built by evolution, rather than design. This also has the advantage of involving the user in the process in a direct and tangible way.

For a relatively simple system, this approach can work. For the average industrial strength application, however, skipping analysis and design will have the effect of lengthening the overall time to produce a viable system rather than shortening it. After all, for creating something as complex as a human being, evolution by natural selection can work – if you have billions of years available.

WHAT TIME CAN BE SAVED?

All right, then, what can we do to shorten the time required to build a system? If we cannot eliminate strategy, analysis and design, can we at least reduce the length of time they require? The answer is yes – at least in some kinds of situations.

The time required to develop a set of strategic data models in an organization can be saved by using standard models. It is still useful to interview the main players to determine priorities, objectives, and constraints, and it is also a good idea to feed the models back to these people to ensure that they understand and agree to the general concepts. But if you don't have to build the models from scratch. If there are no unusual circumstances, you may be able to outline a strategic study in less than a month. You may even be able to establish a framework for the analysis project in less than a week.

When doing detailed analysis, using the standard models as a starting point will save many weeks of modeling effort. Using these models is also protection if a strategy study was not done. If you are addressing order processing without addressing manufacturing, for example, it is comforting to know that the standard order processing model is basically compatible with standard manufacturing environments.

As for the analysis itself, the standard models are sufficient that you could begin the project with a feedback session, although a few key interviews up front are useful for establishing the basic parameters of the operation and its interests.

These models are not inviolate. From the feedback session and the follow-up interviews, lots of variations may be called for. Figure 2, for example, shows an elaboration on the model for a particular client.

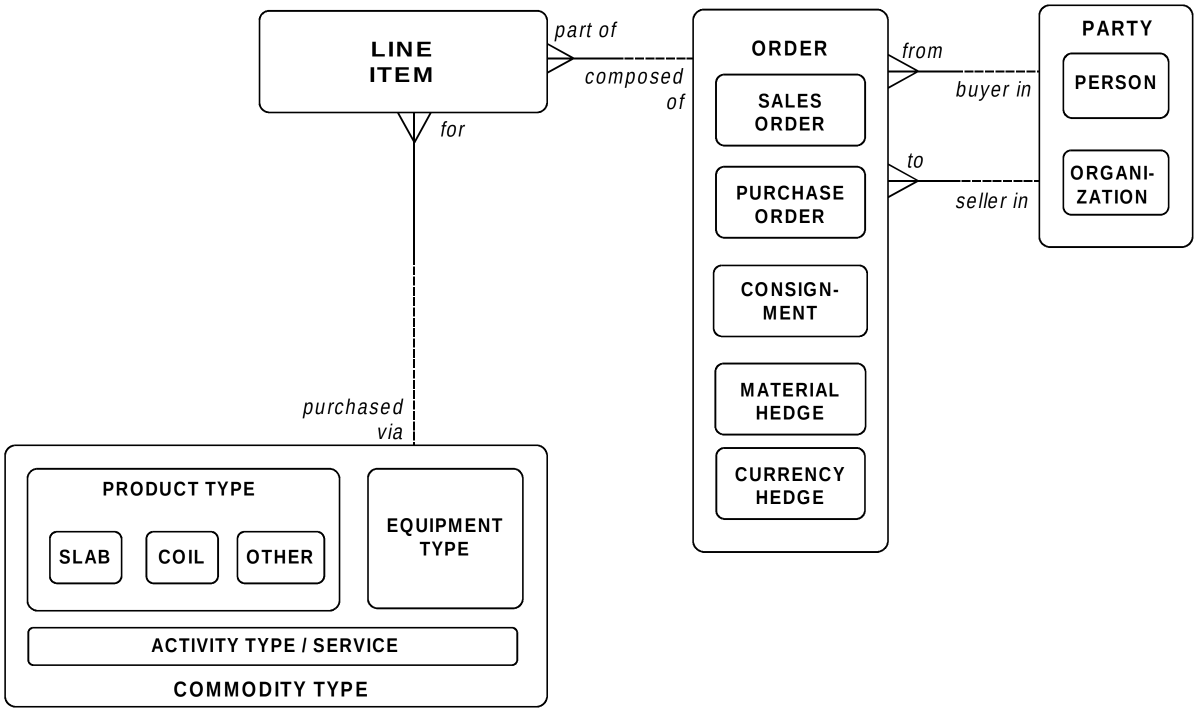

Figure 2: Client's Model

Note that the basic structure survived, but with several twists. This is an aluminum manufacturer, so PRODUCT TYPES are either SLABS of raw aluminum, COILS of finished aluminum, or OTHER materials used in the plant. This is an international client, so it is also true that each ORDER must be *in terms of* one and only one CURRENCY. Moreover,h it turns out that this company hedges currency, since the product may be delivered and paid for many months in the future. A kind of ORDER then, may be a CURRENCY HEDGE which is *for the purchase of*, not a COMMODITY TYPE, but a CURRENCY at a future date. Thus a CURRENCY HEDGE (ORDER) may be *in terms of* Bahraini Dinars, *for the purchase of* Japanese Yen in four months.

Similarly, a MATERIAL HEDGE may be added, which is the purchase of raw materials at a guaranteed price at some point in the future.

A more esoteric variation is for a LETTER OF CREDIT or some other kind of bank guarantee. (See Figure 3.) While this is an exotic extension, even this has the same structure as the sales order whose payment is being guaranteed. Indeed one of the application constraints is that (since each LETTER OF CREDIT must be *to guarantee* one and only one SALES ORDER) the terms and attributes of the LETTER OF CREDIT must be identical to those of the SALES ORDER.

Note that, by starting with a standard model, the discussion immediately focuses on that which is unique to the organization. It is not necessary to reinvent the models from scratch. Moreover, even when dealing with ostensibly new things, it is often possible to take standard structures and extend them to cover them.

If standard data models are possible, what about standard models of the other kinds of things that concern a system developer: functions, locations, people, events, and motivation?

The time required to produce a function model can also be reduced, if you recognize that the structure of many functions reflects the structure of the data. The data maintenance functions, implemented by data entry and retrieval modules, reflect the structure of the data base. Standardization of data structures can, to a large extent, have the effect of standardizing these functions. For example, in Figure 4, you could expect there to be the functions “Enter sales order” and “Define product type”. Other functions involve more complex activities of the business, and these must be defined specifically.

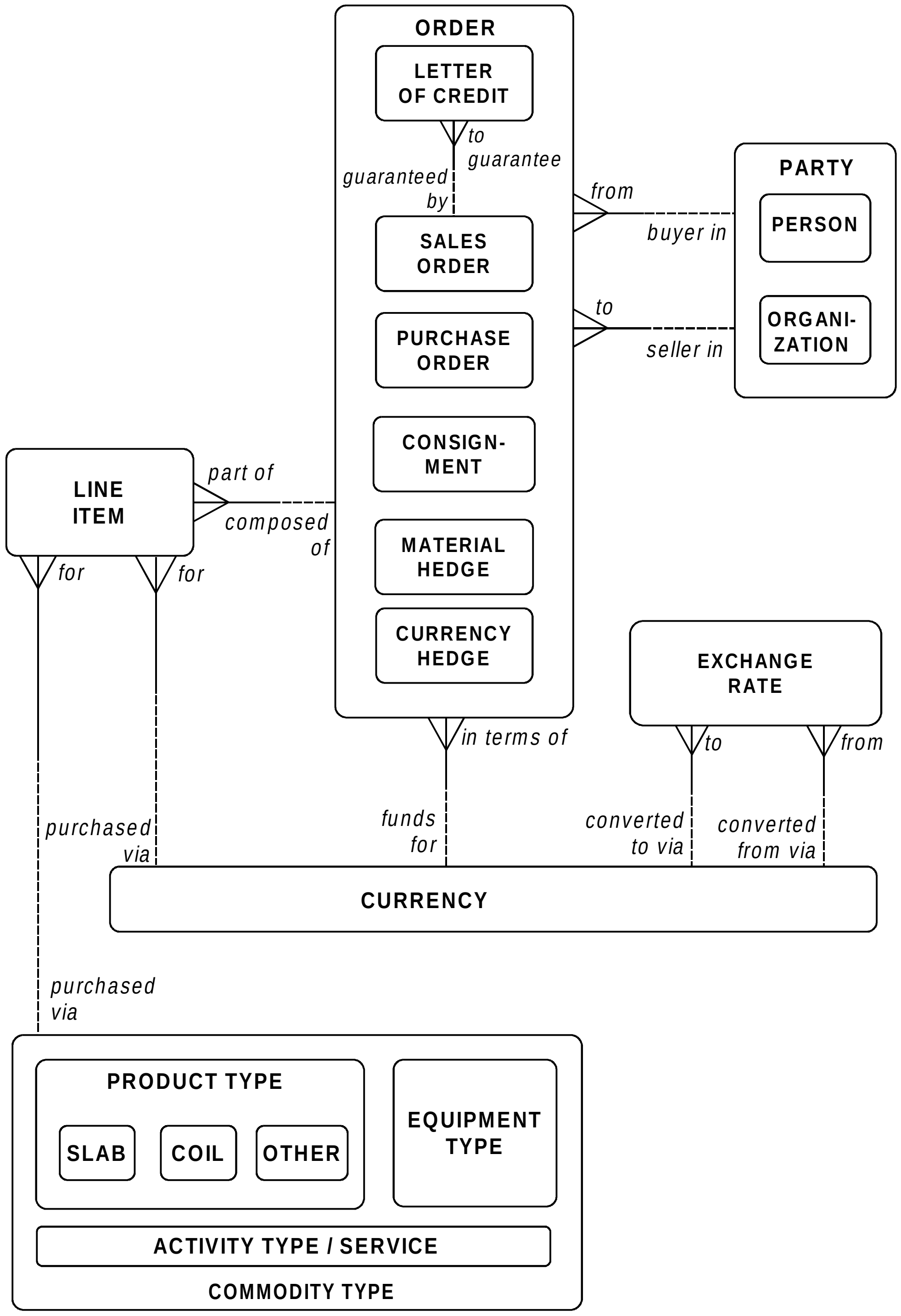

Figure 3: Letter of Credit

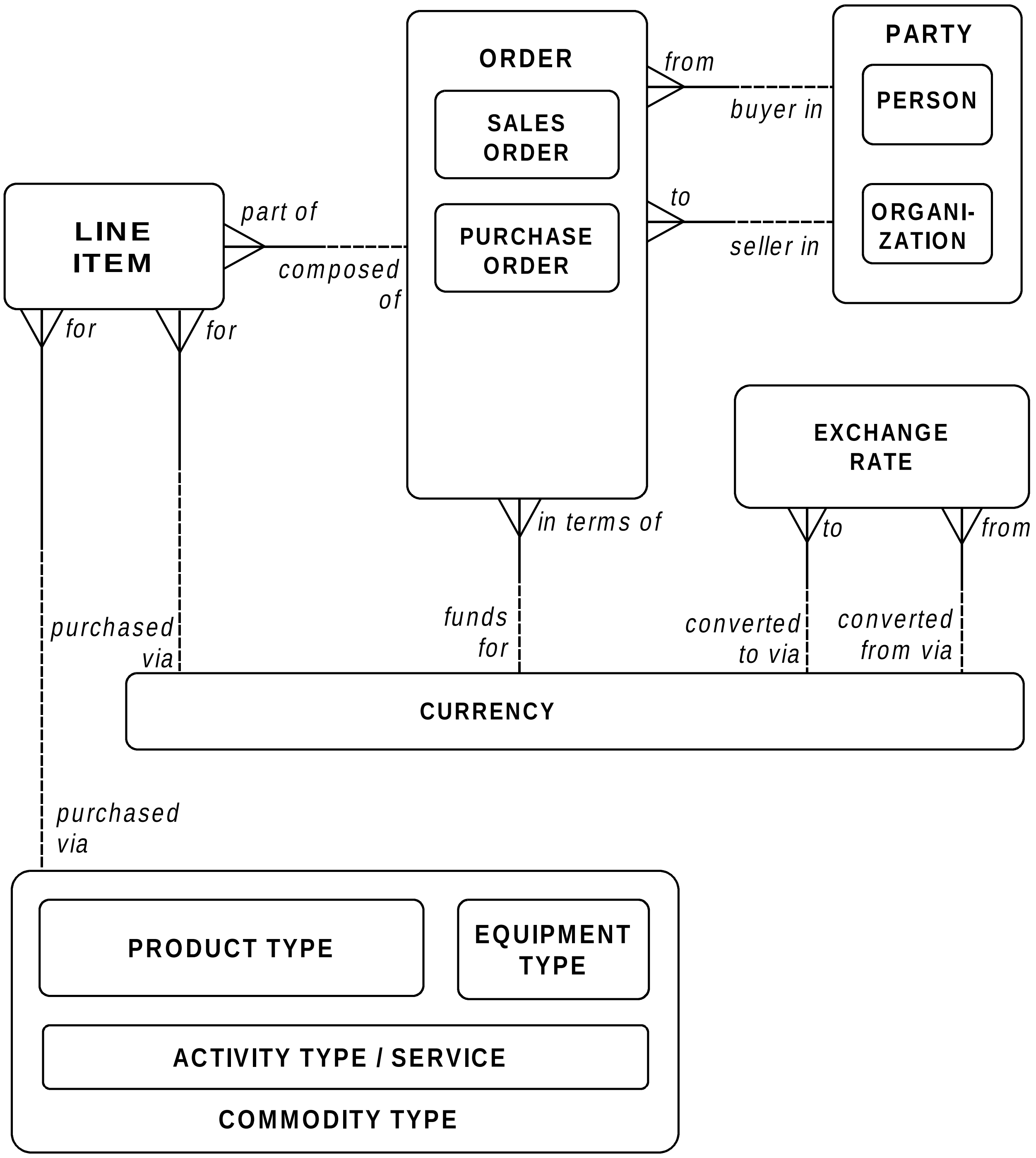

Figure 4: Functions Against the Data Model

The modeling techniques currently available for describing locations and organization structure are relatively primative. Such as they are (organization charts and network diagrams), they can usually be done quickly.

It is not clear whether there are standard patterns for event models.

Constraints, reflecting business rules, will have to be implemented by modules attached to the database or to the data entry structures. It remains to be seen whether there exist anything like "constraint patterns." Some data model structures imply constraints: "must be one . . . ", "may be no more than one . . . ", "must be either one . . . or one . . ." Models which have multiple paths between two entities imply the need for business rules to be expressed, in order to keep the two paths consistent, but there is nothing in the model to say what those rules would be.

In many cases, however, the rules are very specific to the business situation. In Figure 5, the rule on PRODUCT STRUCTURE ELEMENT is that no "loops" are permitted. This is a standard rule that may be invoked whenever there is a "structure" type of entity. On the other hand, the rule that a CURRENCY HEDGE may only be *compose of* LINE ITEMS *for* currencies — is probably specific to this business, since CURRENCY HEDGE was invented specifically for this business. There is nothing in the nature of data model patterns themselves that can make dealing with and implementing constraints any easier. But there is nothing else in RAD to do so either.

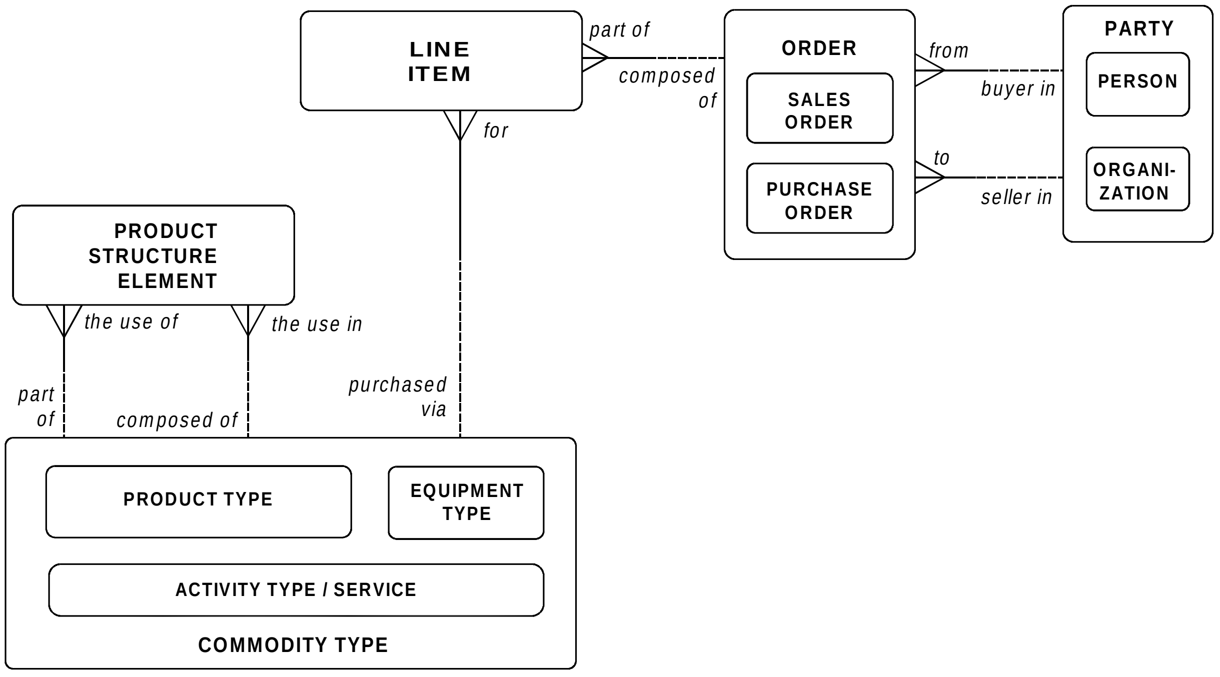

Figure 5: No Patterns in Business Rules

APPROACH

If no more time is available, it is possible to condense the strategy study to publication of a statement of objectives and direction. During analysis, patterns are a very effective way to develop an initial model of the kind of organization you are trying to analyze, so that you can get to a feedback session immediately. Then the bulk of the analyst's time can be devoted solely to dealing with the things that are unique to this organization.

NET EFFECT

Every project is different, and it is impossible to say definitively how much time can be saved by using patterns, but based on the author's experience, it is not unreasonable to expect that strategy studies, the modeling for which could take three to four months, could be done in a month or less. Similarly, requirements analysis projects, which would otherwise be of the magnitude of four to six months could be reduced to one to two months.

The time for design is not shortened by the use of patterns, although CASE tools are effective at quickly translating models to designs. In fact, after the initial prototype, design could be lengthened as designers make decisions to convert generic entities to more concrete tables. (The order entity, for example, becomes separate PURCHASE ORDERS and SALES ORDERS tables.)

So, if you want to develop systems rapidly, don't leave important steps out. Just do them faster. RAD at the expense of thoughtful design is a bad bargain.