A System for Measuring Function Points from an ER-DFD Specification

Evelina Lamma[1], Paola Mello[2], Fabrizio Riguzzi[1*]

[1]Dipartimento di Ingegneria, Università di Ferrara

Via Saragat, 1 44100 Ferrara

{elamma,friguzzi}@ing.unife.it

[2]DEIS - Università di Bologna

Viale Risorgimento, 2 40136 Bologna

pmello@deis.unibo.it

Abstract: We present a tool for measuring the Function Point software metric from the specification of a software system expressed in the form of an Entity Relationship diagram plus a Data Flow Diagram (ER-DFD). First, the informal and general Function Point counting rules are translated into rigorous rules expressing properties of the ER-DFD. Then, the rigorous rules are

* Corresponding author: tel ++39 0532 974836 fax ++39 0532 974870

translated into Prolog. The measures given by the system on a number of case studies are in accordance with those of human experts.

1 Introduction

Software metrics are emerging as a powerful tool for the management of the software development process. Software metrics allow to apply engineering principles to software development, providing a quantitative and objective base for process and technology decisions.

Among software metrics, Function Points [1],[2] (FP) give a measure of the size of a software system by measuring the functionalities that the system offers the user. This metric is applicable both at the beginning of the development process, in the requirements or specification phases, and at the end of the process, after implementation. The importance of FP lies in the fact that they can be used to estimate the cost of a software development project given the requirements or specification because in [2] it has been shown that FP are highly correlated with work-hours.

The rules for counting FP are described in the Function Point Counting Practices Manual [3] published by the International Function Point User Group (IFPUG) [4]. Recently IFPUG FP counting rules have become ISO standard number 20926 [5].

There has been recently a debate about the use of FP [6],[7]. The main problem with FP is that they are not completely independent from the person doing the count. Kemerer [8] reported a 12% difference for the same product by people in the same organization, while Low and Jeffery [9] reported a 30% variance within an organization and more than 30% across different organizations. In order to overcome this problem [7] suggested to simplify the counting rules so that FP can be automatically counted from early system representations.

In the spirit of the suggestion of [7], this paper proposes a system for the automatic counting of FP from a specification of the software expressed in the form of an Entity Relationship (ER) diagram plus a Data Flow Diagram (DFD). The system is called FUN (FUNction point measurement) and appeared in a preliminary version in [10].

In order to develop the system, the informal IFPUG counting rules are specialized for the case of ER-DFD and made rigorous. To achieve this aim, a number of assumptions were done. The informal counting rules expressed in natural language in [3] have thus been translated into rigorous rules expressing properties of the ER-DFD graph. FUN has been tested on seven different case studies and obtained results very close to those of human counters.

The tool is valuable for three reasons. First, it saves human effort. Second, it has been experimentally shown to be in accordance with human counters. Third, if consistently used in an organization or across organizations, it overcomes the problem of dependence from the counter.

There exist in the market a number of commercial tools that help the software engineer in the process of FP counting but none of them is fully automatic [4]. To the best of our knowledge, the only tool that is able to automatically count FP is described in [11] and was developed in academia. Such a system starts from a specification written in UML .

FUN has been implemented in Sicstus Prolog version 3#5 [12] and is available from `http://www.ing.unife.it/software/FUN/`.

The paper is organized as follows. In Section 2 we describe the FP measurement process. In Section 3 we recall ER and DFD specifications and present their integration into ER-DFD specifications. Section 4 describes the application of FP rules to ER-DFD and Section 5 illustrates the system implementation in Prolog. Section 6 shows the application of the system to

a number of case studies. Related works are discussed in Section 7. Conclusions and a discussion of future work follow.

2 Function Point Measurement Process

FP measurement rules are defined in the International Function Point User Group (IFPUG) Counting Practices Manual [3]. The method is based on identifying and counting the *functions* that the system has to provide, i.e. *Internal Logical Files* (ILFs), *External Interface Files* (EIFs) (*data functions*), *External Inputs* (EIs), *External Outputs* (EOs) and *External Inquiries* (EQs) (*transaction functions*). Each function identified in the system is then classified into three levels of complexity (*simple*, *average* and *complex*), and a number of FP is assigned to each function according to its type and complexity. The rules for identifying the functions and for determining their complexity are expressed in natural language and they refer to a number of high level abstractions defined in the manual [3]. Rules have been kept informal and abstract so that they can be applied to any kind of description of the system, from a requirement document to an implementation of the system. However, as a consequence, they are, to a certain extent, vague and not completely free from ambiguities.

The sum of the FP contributions from all the functions gives the unadjusted FP count *UFP*:

$$UFP = \sum_{i \in Types} \sum_{j \in Complexity} w_{ij} x_{ij}$$

where *Types*={ILF, EIF, EI, EO, EQ}, *Complexity*={simple, average, complex}, w_{ij} is the number of Function Points assigned to a function of type i and of complexity j and x_{ij} is the number of functions of type i that have complexity j. The values of w_{ij} are given by the IFPUG manual while the values of x_{ij} are computed by the counter.

The final FP count is then obtained by multiplying the unadjusted count by an adjustment factor that expresses the influence of 14 *general characteristics* of the system on which the application will run.

FP count is thus performed in 6 steps:

1) identifying the type of FP count: development project, enhancement project or application;
2) identifying the boundary of the application subject to the measure;
3) identifying the data functions, classified as Internal Logical Files and External Interface Files, and evaluating their complexity by counting the number of Data Element Types (DET) and Record Element Types (RET) for each function;
4) identifying the transaction functions, classified as External Inputs, External Outputs and External Inquiries and evaluating their complexity by counting the number of Data Element Types (DET) and File Types Referenced (FTR) for each function;
5) determining the number of unadjusted FP by summing the contributions of all functions;
6) computing the final number of FP by multiplying the unadjusted FP count by the adjustment factor.

Our aim is to automate steps 3), 4) and 5) starting from the specification of the system in terms of an ER-DFD diagram. Steps 3) and 4) are the most complex, time consuming and prone to error, therefore they are the most interesting to automate. We assume to be performing a development project count and that the boundary of the application is indicated in the ER-DFD diagram. We do not automate step 6 because it requires many notions on the application and on the environment that are not present in the ER-DFD specification and are difficult to formalize. Moreover, many companies prefer to consider only the unadjusted FP count and the ISO has standardized only the unadjusted FP count

3 Entity Relationship - Data Flow Diagrams

We will perform the measurement on the specification of the application expressed by an Entity Relationship diagram [13] integrated with a Data Flow Diagram [14]. We consider an integration of the diagrams which is similar to Formal Data Flows Diagrams [15]: the data stores of DFD are replaced by entities and relationships of the ER diagrams, therefore we have data flows entering directly into entities and relationships and data flows coming out from them. Moreover, we distinguish three types of data flows: proper data flows, that represent the exchange of data; error flows, that represent the exchange of error messages, and control flows, that represent the exchange of control information. We call such an integrated diagram an ER-DFD. In order to distinguish between elements of the DFD and ER diagram which have a similar graphical symbol, we adopt the following conventions: external agents (the user or other applications) of DFD are represented with a dashed line box to distinguish it from entities represented as normal boxes. Data flows, error flows and control flows are represented by arrows to distinguish them from the connections between entities and relationship represented as simple lines. Data flows, error flows and control flows are distinguished on the basis of the line of the arrow: continuous for data flows, dashed for error flows and dotted for control flows. Figure 1 shows a sample diagram. Note that either the symbol "1" or "M" (for many) is associated to each connection between an entity and a relationship to represent the functionality of a relationship: for example, in figure 1 relationship rel1 is one to many from entity2 to entity1, meaning that an occurrence of entity1 appears in the relationship at most 1 time and an occurrences of entity2 appear in the relationship at most M times.

A number of fields are associated with each data flow: when a field has the same name of the attribute of an entity, they refer to the same data. When the field does not correspond to any

attribute, it represents data derived from attributes by computation. Error flows and control flows do not have any fields associated with them.

We suppose that the diagram contains also the indication of the boundaries of the different applications in the form of dashed lines.

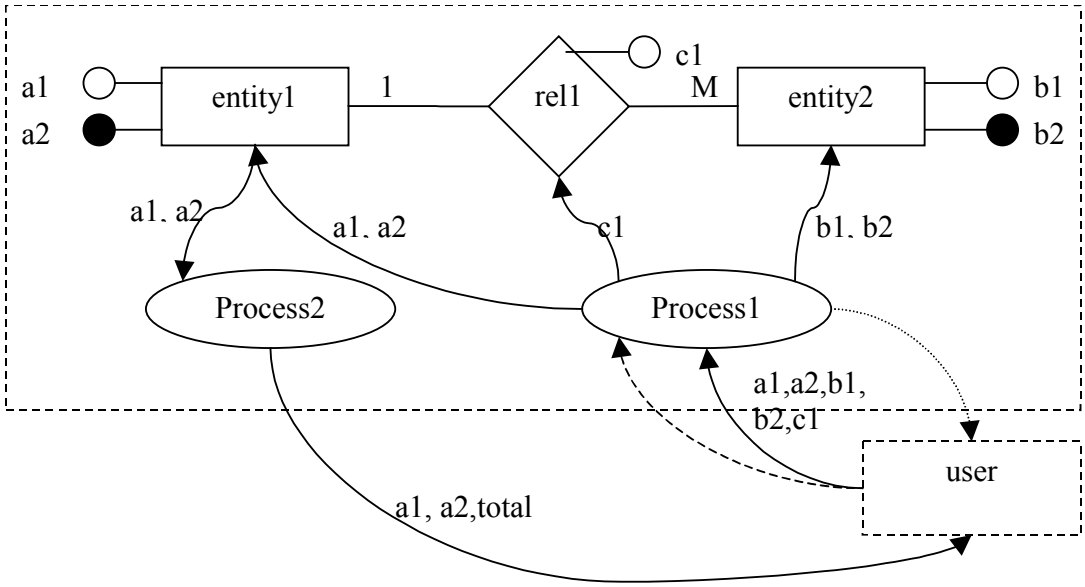

Figure 1: Example of an ER-DFD diagram.

In figure 1, two processes are shown. As can be seen, the two processes can exchange data by using an element of the ER diagram: in this example, the data that is read by Process2 from entity1 is written by Process1.

In order to perform the count, a number of assumption on the ER-DFD graph have been made. We assume that every attribute of an entity or a relationship is unique, user recognizable and non-repeated in the sense of [3]. This assumption will be clarified in section 4.2.

Each process in the DFD must be an elementary process in the sense of [3]:

"An elementary process *is the smallest unit of activity that is meaningful to the user(s). .... The elementary process must be self-contained and leave the business of the application being counted in a consistent state."* [3].

Assuming that every process in the DFD is an elementary process is reasonable if we suppose that the DFD processes reflect the basic activities to be performed by the application. We assume also that the processing logic of every process in the DFD is unique. This assumption is not very restrictive since it is reasonable for a DFD diagram not to have duplicated processes.

4 Rules for Counting Function Points from ER-DFD

In this section we present the rules for counting Function Points from the specification of an application expressed in the form of an ER-DFD. For each function, we have translated IFPUG informal rules into rigorous rules expressing properties of the ER-DFD graph. The rules obtained are rigorous because all the ambiguities and vagueness of IFPUG rules have been removed. It was thus easy to translate them into code. In order to remove ambiguities and vagueness we have interpreted IFPUG rules in the way we thought was more reasonable. To do so we had to make a number of assumption that are listed in section 3 and are also reported below.

In Sections 4.1 and 4.2 we discuss identification and complexity rules for data functions. In Sections 4.3 and 4.4 we describe identification and complexity rules for transaction functions.

4.1 Identification Rules for Data Functions

Data functions are Internal Logical Files and External Interface Files.

4.1.1 Internal Logical Files

The IFPUG definition of an ILF is:

"An internal logical file (ILF) is a user identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted." [3].

Let us now introduce the terminology used in IFPUG's manual.

*"*Control Information *is data that influences an elementary process of the application being counted. It specifies what, when, or how data is to be processed."* [3].

"The term user identifiable *refers to defined requirements for processes and/or groups of data that are agreed upon, and understood by, both the user(s) and software developer(s)."* [3].

"The term maintained *is the ability to modify data through an elementary process."* [3].

The IFPUG identification rule for ILFs is: a group of data or control information is an ILF if it satisfies all the following conditions:

"• The group of data or control information is logical and user identifiable.

• The group of data is maintained through an elementary process within the application boundary being counted" [3].

When applying the above rule to ER-DFD, groups of logically related data are represented by sets of connected entities and relationships while elementary processes are represented by processes of the DFD. As discussed in section 3, we have made the assumption that every process in the DFD is an elementary process.

We need also the following definition:

Definition 1 (Maintained) An entity or a relationship is maintained by a process if and only if there is a data flow from the process to the entity or relationship. A set of entities and

relationships is maintained by a process if and only if the process maintains all the elements of the set.

We can present now the ILF identification rule for ER-DFD.

Rule 1 (ILF identification) A set F of entities and relationships is an ILF if:

1) **all the elements of the set are inside the application boundary;**
2) **there is a single process of the application that maintains the set and no elements outside it;**
3) **the entities and relationships of F are connected;**
4) **no proper subset of F is an ILFs.**

The fourth condition above is required in the case in which a process maintains a set of entities and relationships that properly contains at least one ILF: in that case the set is not counted as an ILF because the process is not the simplest one that maintains elements of the set. In order to test condition 4, the system has to consider all possible subsets of F and to check that none of them is an ILF.

In the following we show some examples of ILFs. Figure 2 shows the simplest case: an ILF composed by a single entity. In this case the entity is clearly modified through an elementary process of the application and therefore forms an ILF by itself.

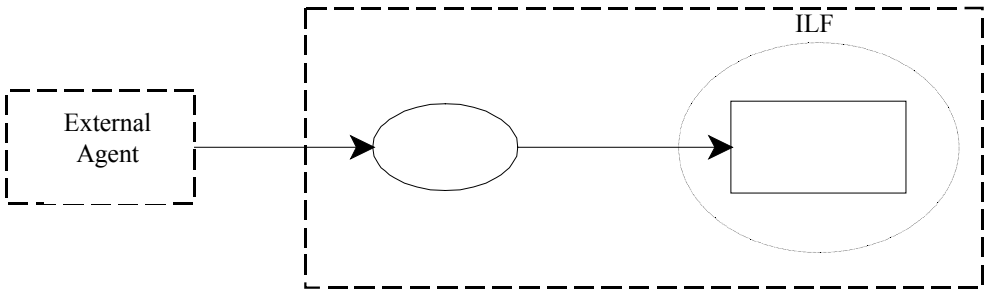

Figure 2: An ILF composed of one entity

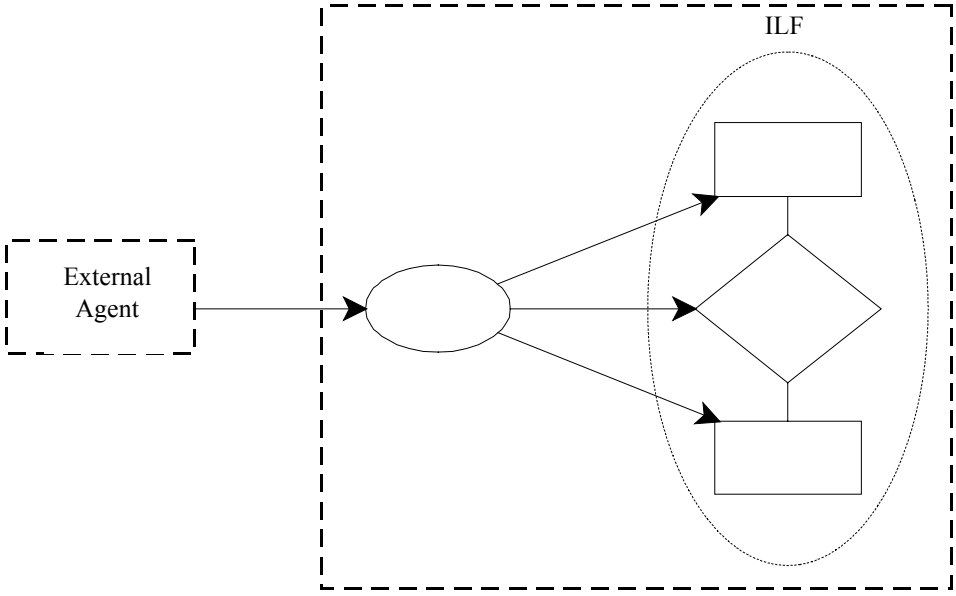

Figure 3: An ILF composed of two entities and one relationship.

Figure 3 shows an ILF composed by two entities and the relationship connecting them. Each entity separately would not be an ILF because an elementary process does not maintain them. In fact, modifying just one entity does not leave the application in a consistent state. Both entities and the relationship between them must be updated to reach a new consistent state; therefore they form a single ILF.

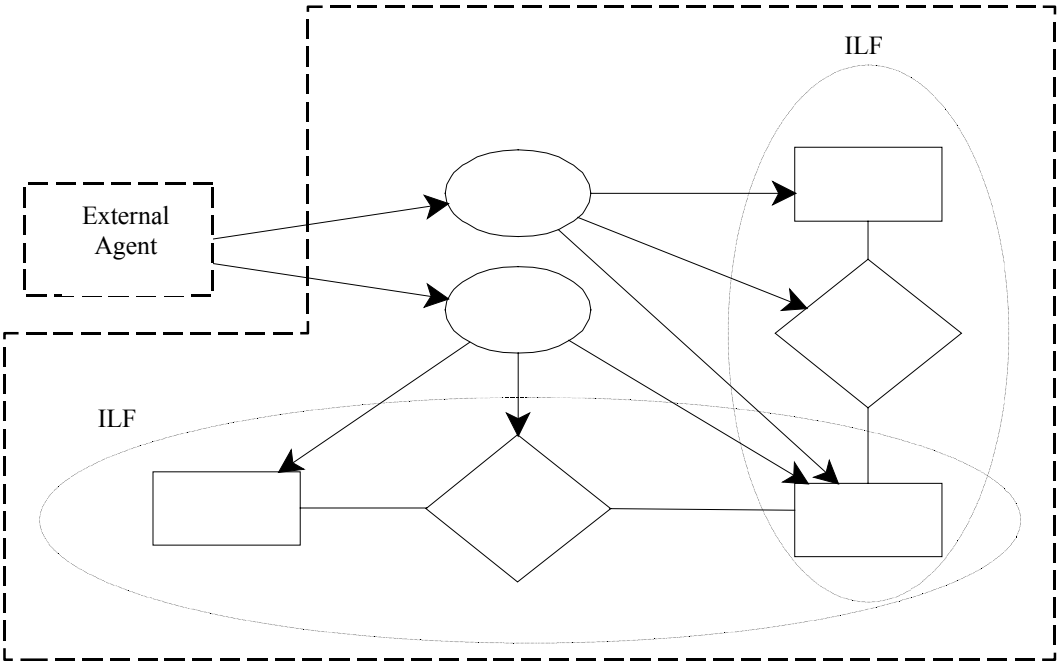

Figure 4: Two ILFs composed of two entities and one relationship.

Figure 4 presents a more complex case: two ILFs that share an entity. It may seem unnatural to have the same entity in two different ILFs. However, if we consider the common entity as an ILF by itself, we contradict the condition that an ILF is maintained through an elementary process of the application, because no process maintains that entity alone.

4.1.2 External Interface Files

The IFPUG definition of an EIF is:

"An external interface file (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one

or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application." [3].

The IFPUG identification rule for EIFs is: a group of data or control information is an EIF if it satisfies all the following conditions:

"• The group of data or control information is logical and user identifiable.

• The group of data is referenced by, and external to, the application being counted.

• The group of data is not maintained *by the application being counted.*

• The group of data is maintained in an ILF of another application." [3].

We first give a definition for an entity or a relationship referenced by a process.

Definition 2 (Referenced) An entity or a relationship is referenced by a process if and only if there is a data flow from the entity or relationship to the process. A set of entities and relationships is referenced by a process if at least one of the members of the set is referenced by the process.

We can present now the EIF identification rule for ER-DFD.

Rule 2 (EIF identification) A set of connected entities and relationships is an EIF if:

1) all the elements of the set are inside an external application;

2) the set satisfies the condition for ILFs with respect to the external application;

3) it is referenced by a process inside the counted application;

4) the set is not maintained by any process inside the counted application.

In Figure 5 we show an example of an EIF composed of two entities and a relationship. The set is outside the counted application boundary and satisfies the ILF rule for the external application. Moreover, it has a data flow going from one of its entities to a process in the counted

application: it is important to note that it is not necessary to have data flows from all the elements of the set. Finally, no data flow enters in the set from the counted application.

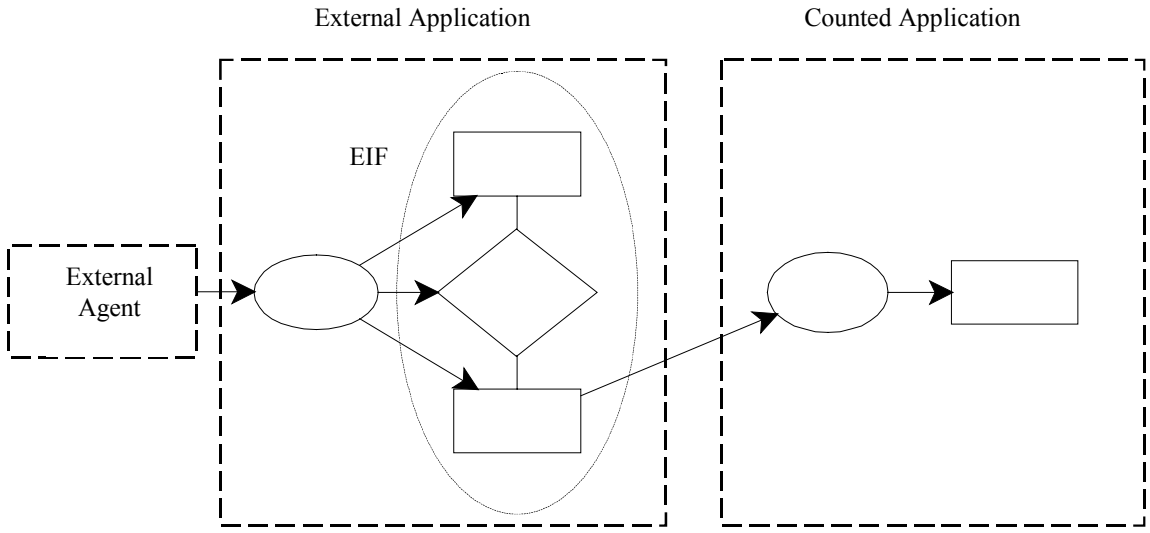

Figure 5: Example of EIF.

4.2 Complexity Rules for Data Functions

In order to assign the right number of FP to each identified data function, we have to count the Data Element Types and Record Element Types associated with the function.

"A data element type *is a unique user recognizable, non-repeated field."* [3].

We make the assumption that every attribute of an entity or a relationship is unique, user recognizable and non-repeated, i.e., it is a DET. This assumption is not restrictive, if the ER diagram has been correctly laid down.

Moreover, the IFPUG manual requires to count as DETs each piece of data that is required to establish a relationship with other logical files. Therefore, we count as DETs also the attributes in the key of external entities connected to a relationship inside the file.

Rule 3 (DET counting) Count:

1) **one DET for each attribute of the entities and relationships in the logical file;**
2) **one DET for each attribute composing the key of each entity in other files connected to a relationship belonging to the file.**

In order to count DETs, each entity and relationship of the logical file is considered in turn, and DETs are counted for it. Then all counted DETs are summed to give the total for the file.

The IFPUG definition for RETs is:

"A Record Element Type (RET) is a user recognizable subgroup of data elements within an ILF or EIF" [3].

The IFPUG counting rule is:

"Count a RET for each optional or mandatory subgroup of an ILF or EIF or if there are no subgroups, count the ILF or EIF as one RET" [3].

When the user adds data regarding a new instance to an ILF or EIF, she/he must add at least one of the *mandatory subgroups* and zero, one or all *optional subgroups*. If an entity does not have sub-entities, then it corresponds to a single subgroup. If it has sub-entities, then a subgroup consists of the parent entity combined with one or more sub-entities, depending on the type of hierarchy. In order to illustrate how subgroups are composed, let us consider the case of an entity with two sub-entities. We have four cases.

1) Exclusive and total hierarchy: we have 2 mandatory RETs, one consisting of the parent entity plus one sub-entity and the other consisting of the parent plus the other sub-entity. For example, the entity "person" has two sub-entities, "male" and "female". A "person" is either a "male" or a "female" but not both, so the hierarchy is exclusive and total. Therefore we have two RETs, one for male persons and one for female persons.

2) Exclusive and not total hierarchy: we count 3 RETs, 1 mandatory for the parent entity and 2 optional for each of the sub-entities, because an instance of the parent may not be in any of the children. For example, the entity "vehicle" has sub-entities "car" and "bike". A "vehicle" can be a "car" or a "bike" but not both. Moreover, a "vehicle" can be also something else, therefore the hierarchy is exclusive and not total. In this case we have 3 RETs, one for cars, one for bikes and one for vehicles that are neither cars nor bikes.
3) Not exclusive and total hierarchy: we count 3 mandatory RETs, one for the parent entity plus one child, one for the parent entity plus the other child and one for the parent entity plus both children. For example, the entity "boat" has sub-entities "sea boat", "inland water boat". The hierarchy is total because it considers all kinds of boats but is not exclusive because there are boats that travel both on the sea and on inland water. Therefore we must count 3 RETs: one for sea boats, one for inland water boats and one for boats that travel on both the sea and inland water.
4) Neither total nor exclusive hierarchy: we count 1 mandatory RET for the parent and 2 optional RETs for the children. For example, the entity "person" has sub-entities "student" and "worker". There exist persons that are neither workers nor students (e.g. retired) and a person can be both a student and a worker, therefore the hierarchy is neither total nor exclusive. In this case we have three RETs, one describing persons, one describing persons that are students and one describing persons that are workers.

We can now give the rule for counting RETs on ER-DFDs.

Rule 4 (RET counting) Count one RET for each entity and for each relationship with attributes of the logical file. If an entity has sub-entities, then count RETs for each sub-

entity, sum the results over all sub-entities and add 1 if the hierarchy is not both total and exclusive.

As for DETs, we count RETs for each entity and relationship separately and then we sum the results.

Once the number of DETs and RETs for ILFs and EIFs have been determined, it is possible to compute the number of function points to be assigned to each function. For each function type, a complexity matrix gives the function complexity given the number of DETs and RETs. The complexity can take the values: low, average or high. From the value of the complexity, a conversion table gives the number of function points. Table 1 shows the *complexity matrix* for ILFs and EIFs, Table 2 shows the conversion table for ILFs and Table 3 the conversion table for EIFs.

	1-19 DET	20-50 DET	51 or more DET
1 RET	low	low	average
2-5 RET	low	average	high
6 or more RET	average	high	high

Table 1: Complexity matrix for ILFs and EIFs.

Complexity	Function Points
low	7
average	10
high	15

Table 2: Conversion table for ILFs.

Complexity	Function Points
low	5
average	7
high	10

Table 3: Conversion table for EIFs.

4.3 Identification Rules for Transactional Functions

Transactional Function are classified into External Inputs, External Outputs and External Inquiry.

4.3.1 External Inputs

"An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system." [3].

We distinguish between EI of data and EI of control information. The IFPUG rule states that the conditions that must be satisfied by a process to be an EI are:

"• The data or control information is received from outside the application boundary.

- *At least one ILF is maintained if the data entering the boundary is not control information that alters the behavior of the system.*
- *For the identified process, one of the following three statements must apply:*
  - *Processing logic is unique from the processing logic performed by other external inputs for the application.*

- *The set of data elements identified is different from the sets identified for other external inputs for the application.*
- *The ILFs or EIFs referenced are different from the files referenced by other external inputs in the application."* [3].

The simplest case of EI of data is shown in Figure 2.

The simplest case of an EI of control information is shown in Figure 6.

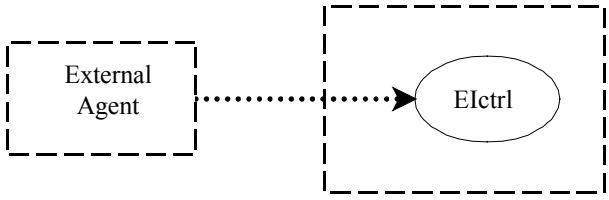

Figure 6: Simplest case of EI of control information.

In order to identify EIs in the ER-DFD diagram, we have to consider the processes in the diagram. We have assumed in section 3 that the processing logic of every process in the DFD is unique.

Rule 5 (EI identification) A process of the ER-DFD belonging to the application is an EI if

1) there is at least one data flow from the outside to the process,

2) the process maintains at least one element of an ILF

or

1) there is a flow of control information from the outside to the process.

4.3.2 External Inquiry

"An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present

information to a user through the retrieval of data or control information from an ILF of EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered." [3].

Figure 7 shows the simplest case of EQ.

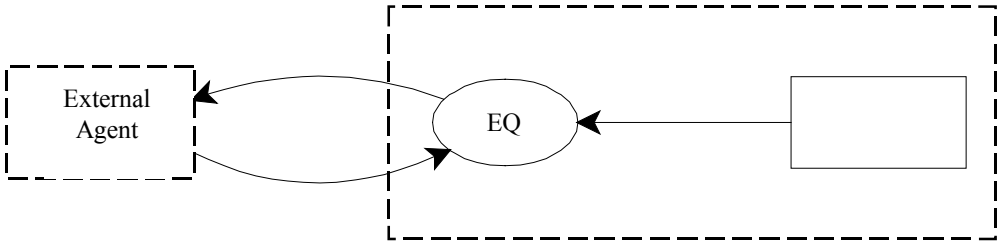

Figure 7: Simplest case of an EQ.

Rule 6 (EQ identification) A process of the ER-DFD is an EQ if

1) **there is at least one data flow from the outside to the process;**
2) **there is at least one data flow from the process to the outside;**
3) **there is at least one data flow from an element of a file to the process;**
4) **all the fields of data flows going outside the application boundary are among the fields of data flows to the process;**
5) **no elements of an ILF is maintained by the process.**

4.3.3 External Output

"An external output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information . The processing logic must contain at least one mathematical

formula or calculation, create derived data, maintain one or more ILFs or alter the behavior of the system." [3].

Figure 8 shows the simplest case of an EO.

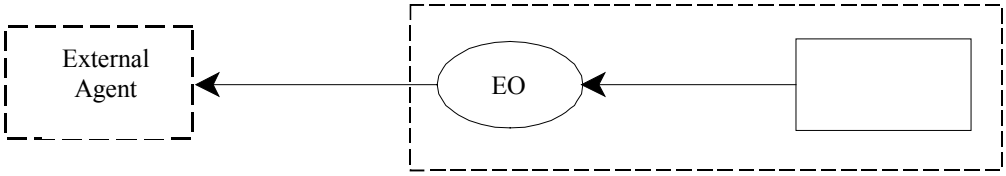

Figure 8: Simplest case of EO.

What distinguishes an EO from an EQ is the fact that an EQ does not elaborate the retrieved data, while an EO outputs derived data. Therefore we have the following rule for ER-DFD.

Rule 7 (EO identification) A process in the ER-DFD is an EO if

1) there is at least one data flow from a file to the process;

2) there is at least one data flow from the process to the outside;

3) data flows from the process to the outside contain at least one field that is not contained in any of the data flows from ILFs to the process.

4.4 Complexity Rules for Transactional Function

In order to assign the right number of FP to each identified transactional function, we have to count the DETs and File Types Referenced (FTRs) associated with the function. The definition for DETs is the same as the one for data functions. The counting rule for DETs of EI is:

Rule 8 (EI, DET counting) Count one DET for each field in data flows from external sources to the EI. Count one DET for each error flow from the EI.

The IFPUG definition of a FTR is:

"A File Type Referenced is

- *an Internal Logical File read or maintained by a transactional function,*
- *an External Interface File read by a transactional function."* [3].

Recalling the definition of a set of entities and relationships (and hence of a file) maintained given in section 4.1.1. and referenced given in section 4.1.2, we can now give the counting rule for FTRs:

Rule 9 (EI, FTR counting) Count one FTR for each ILF maintained or referenced by the process and one FTR for each EIF referenced by the process.

Note that we have interpreted the term "read" in the IFPUG definition of a FTR as the term referenced.

Rule 10 (EQ, EO, DET counting) Count one DET for each field in the data flows from the outside to the process or from the process to the outside plus one DET for each error flow.

Rule 11 (EQ, FTR counting) Count one FTR for each ILF that has at least one DET among the fields of the data flow from the outside to the EQ.

Rule 12 (EO, FTRs counting) Count one FTR for each ILF that is referenced or maintained by the process and one FTR for each EIF that is referenced by the process.

As for data functions types, once the number of DETs and FTRs for a transaction function have been determined, it is possible to compute its function points. First, the complexity of the functions is determined using complexity matrixes. Then, the complexity is translated into function points by means of the conversion tables for EIs, Eos and EQs. For complexity matrixes and conversion tables for transaction functions see [3].

5 Implementation

In this section we first discuss the reasons for which Prolog was chosen for the implementation. Then we describe how the ER-DFD is given as input to FUN and we provide an example of how the counting rules have been translated into Prolog by showing the code that identifies ILFs.

5.1 The Prolog Language

Prolog is a programming language centered around a small set of basic mechanisms, including pattern matching (properly, unification), tree-based data structuring, and automatic backtracking.

Prolog is especially suited for problems involving structured objects and relations between them (see [16] for several applications of Prolog in the Artificial Intelligence domain).

Prolog predicates provide an effective representation for relations, which can be accessed in several ways thanks to the capability of querying a Prolog program in different ways.

Furthermore, whereas conventional languages are procedurally oriented, Prolog adopts a declarative view but maintains, at the same time, a procedural reading of clauses. In this respect, the Prolog implementation of identification and counting FP rules outlined in the following took great advantage from the declarative reading offered by this language.

A Prolog program is composed by a set of definite clauses (i.e., facts and rules), and a Prolog computation is started by querying the program with a goal to be proved.

Let us consider as a simple example the following program, which represents arcs between nodes in a graph and the connected relation (uppercase identifiers are variables):

```
arc(a,b).
arc(b,c).
connected(X,Y):-arc(X,Y).
```

```
connected(X,Y):-arc(X,Z),connected(Z,Y).
```

The goal:

```
:-connected(a,c).
```

succeeds, since the (backward) interpreter is able to prove the corresponding query from the previous set of clauses. But the same program can be queried differently, for instance to ask the goal:

```
:-connected(X,c).
```

in order to find whether there exists a node which is connected to node `c`. This goal succeeds too with answer `X=b`. In backtracking other answers are possibly generated (answer `X=a`).

The same program can also be queried with the goal:

```
:-connected(X,Y).
```

in order to find all the couples `X,Y` where `X` is connected to `Y`. This goal has several answers, too, namely `X=a, Y=b`; or `X=a, Y=c`; or `X=b, Y=c`.

Special-purpose Prolog predicates (e.g., `setof`, `findall`) are available that are able to collect into lists all the objects that satisfy some query. For instance, for the previous Prolog program, the goal:

```
:-findall(X,connected(X,c),L).
```

returns in `L` the list of elements connected to node `c`, namely `L=[b,c]`. This facility is extensively exploited in the Prolog implementation of FP rules.

5.2 ER-DFD Representation

We represent an ER-DFD with the following syntax.

```
application(name,[ent1,...,entn],[rel1,...,relm],[proc1,...,
procp]).
```

denotes that `name` is an application containing entities `ent1,...,entn`, relationships `rel1,...,relm` and processes `proc1, ...,procp`.

We consider the user as an application that contain the only process `user`. Therefore, each ER-DFD description must contain the fact

```
application(user,[],[],[user]).
```

Each entity of an ER-DFD is represented by a fact of the form:

```
entity(name,[key1,...,keyn],[attrib1,...,attribm]).
```

where `name` is an entity with attributes `key1, ...,keyn` as the key and `attrib1, ...,attribn` as non-key attributes.

Hierarchies are represented with Prolog facts of the form:

```
specialization(name,[child1,...,childn],total,exclusive).
```

when entity `name` has sub-entities `child1,...,childn`. Arguments `total` and `exclusive` have to be replaced by the Boolean constants `0` and `1` stating whether the hierarchy is total and/or exclusive. Each subentity must then be represented by the fact

```
subentity(name, [attrib1,...,attribm]).
```

stating that sub-entity `name` has attributes `[attrib1,...,attribm]`.

A relationship between entities is mapped into a Prolog fact of the form:

```
relationship(name, ent1, ent2,[attrib1, ..., attribn], card1,
card2).
```

where `name` is a binary relationship between `ent1` and `ent2` with attributes `attrib1,..., attribn` and cardinality `card1` from `ent1` and `card2` to `ent2`. `card1` and `card2` can assume the values `1` or `m`. The cardinality of the relationship is actually never used by the rules

described above but we have decided to store this information for possible extensions of the current system.

A data flow from `sour` to `dest` with fields `field1,...,fieldn` is represented by a Prolog fact of the form:

```
dataflow(sour,dest,[field1,...,fieldn]).
```

A control flow from `sour` to `dest` is represented by

```
controlflow(sour,dest).
```

and an error flow from `sour` to `dest` is represented by

```
errorflow(sour,dest).
```

As an example, the ER-DFD reported in Figure 1 is represented as:

```
application(appli,[entity1,entity2],[relationship1],
  [process1,process2]).
application(user,[],[],[user]).
entity(entity1,[a1],[a2]).
entity(entity2,[b1],[b2]).
relationship(relationship1,entity1,entity2,[c1],1,m).
dataflow(user,process1,[info]).
dataflow(process1,relationship1,[c1]).
dataflow(process1,entity1,[a1,a2]).
dataflow(process1,entity2,[b1,b2]).
dataflow(entity1,process2,[a1,a2]).
dataflow(process2,user,[a1,a2,total]).
controlflow(user,process1).
errorflow(process1,user).
```

5.3 Example of Translation of the Counting Rules into Prolog

In this section we show how the rule for identifying ILFs has been translated into Prolog. The predicate `ilf/2` is used to identify ILFs, according to Rule 1 and Definition 1 presented in section 4.1.1: `ilf(Appli,ILF)` succeeds if `ILF` is a list containing the entities and relationships of an ILF for the application `Appli`. The ILF identification rule is implemented by the following Prolog clause (in Sicstus Prolog syntax):

```
ilf(Appli,ILF):-
    application(Appli,EntList,RelList,ProcList),
% pick a process Proc of Appli
    member(Proc,ProcList),
    append(EntList,RelList,ERList),
% find the ent. and rel. that are maintained by Proc
    findall(ER,dataflow(Proc,ER,_),ILF1),
    delete(ILF1,user,ILF2),
    remove_duplicates(ILF2,ILF),
% verify that ILF is inside the boundary of Appli,
    subset(ILF,ERList),
% that is not empty,
    ILF \== [],
% that is connected
    connected(ILF),
% that no partitions are ILF
    \+ some_partitions_are_ILFs(Appli,ILF).
```

We use the Sicstus built-in predicate `findall(Template,Goal,Bag)` that assigns to `Bag` a list of instances of `Template` returned by each proof of `Goal` found by Prolog. All variables in `Goal` are taken as being existentially quantified. The call `findall(ER, dataflow(Proc, ER,_), ILF)` returns in `ILF` the list of all the entities and relationships maintained by the process `Proc`. The predicate `subset(Sublist,List)` verifies that all elements of `Sublist` are in `List`.

6 Case Studies

In this section, we describe the application of FUN to a number of software systems. The first is an application for the management of Human Resources that is the subject of a series of case studies [17],[18],[19] of Function Point measurement published by IFPUG. We will here consider [18] in which measurement is performed starting from the specification of the application expressed as an ER diagram and a DFD.

The aim of the application is to manage information about employees of a firm. In particular, the user requires storing information about each employee, including data on the dependents of the employee, data on the salary or the hourly rate and data on the work location. The location must be a valid location in the application Fixed Asset. If the employee works abroad, the hourly rate must be converted to US dollars by accessing the application Currency and retrieving the conversion rate. Moreover, the application has to store information about different jobs. Finally, the user requires to store information about the assignment of jobs to employees. Table 4 shows the attributes of entities and relationships and Figure 9 the ER diagram.

The processes that the user requires are adding, changing, inquiring and reporting information about employees, jobs and job assignments. In inquiring, the user asks for the information regarding an employee, a job or a job assignment given, respectively, a Social_Security_Number, a Job_Number or a Social_Security_Number and a Job_Number. In reporting, the application

Entities or relationships	**Attributes**
Employee	Social_Security_Number (key), Name, Nbr_Dependents, Type_Code.
Salaried	Supervisory_level
Hourly	Standard_Hourly_Rate, US_Hourly_Rate, Collective_Bargaining_Unit_Number
Dependent	Dep_SSN (key), Dep_name, Dep_birth_date
Job	Name, Job_Number (key), Pay_grade
Description	Line_Number, Description_Line
Job Assignment	Effective_Date, Salary, Performance_Rating, Status_Inactive, System_Date
Location	Location_Name (key), Address, City, State, Zip, Country.
Currency	Currency_Location (key), Base_Currency, Conversion_Rate_To_Base_Currency, Date_Of_Rate.

Table 4: Attributes of entities and relationships.

prints a list of all employees, jobs or job assignment together with the total number of shown entities. Moreover, the application should also allow inquiring on location information (process Inquire Locations): it should print, for each employee, its Social Security Number together with the information on the location where he is working.

Among these processes, we will here consider in more details the following:

- adding an employee, together with data on his dependents and the salary or hourly wage (see Figure 10 for the addition process);
- reporting on all employees, printing the list of employees together with their total number (Figure 11);
- inquiring on the data of an employee, given his social security number (Figure 12);
- adding job information, together with its description (Figure 13);
- adding a job assignment (Figure 14).

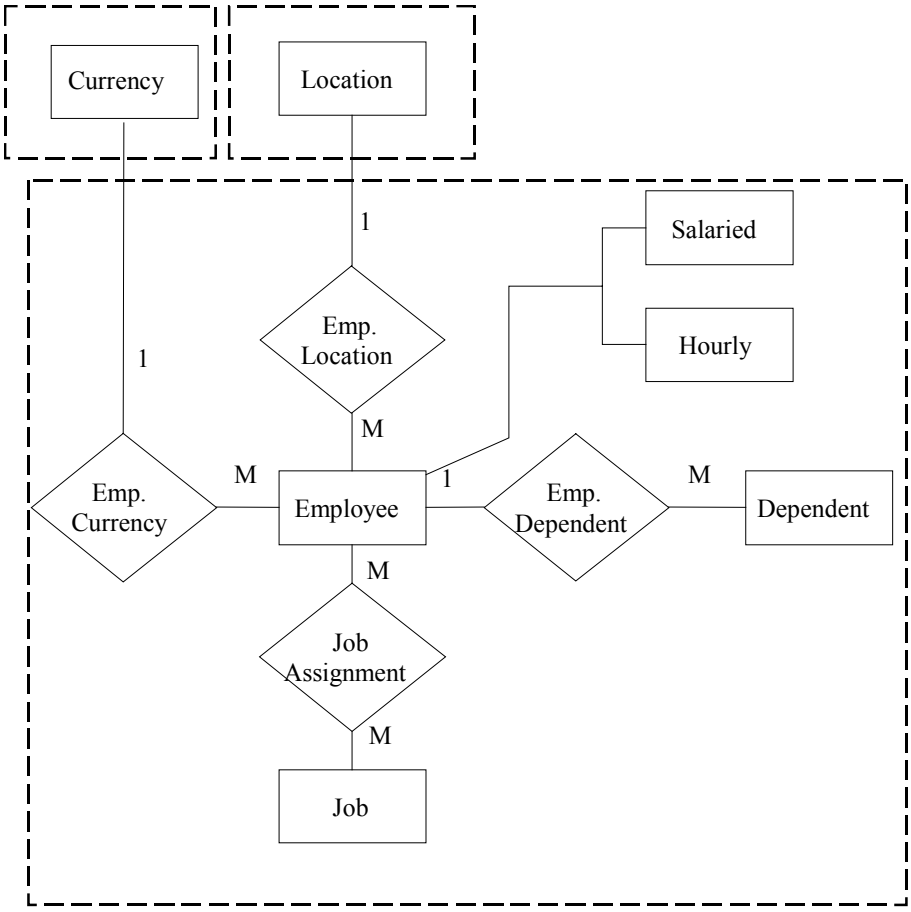

Figure 9: Complete ER diagram for the Human Resource application[1].

[1] This diagram differs from the one in [19] because the relationship Emp. Currency here is inside the application boundary. We believe this diagram is more correct since the relationship Emp. Currency is maintained by processes inside the application.

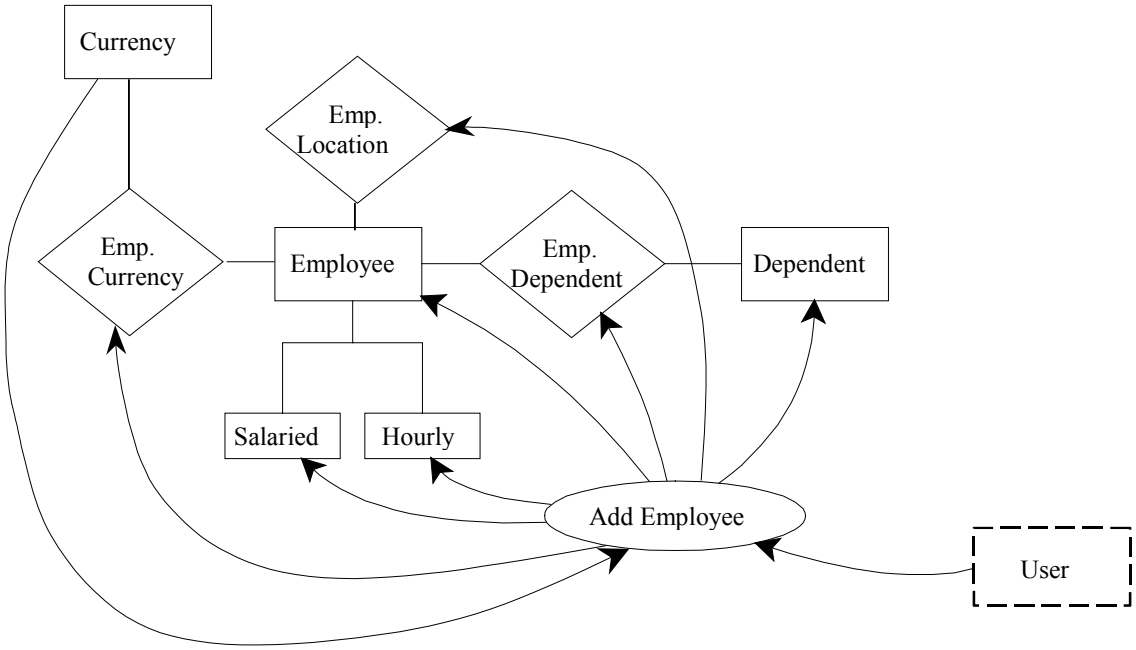

Figure 10: Add Employee process.

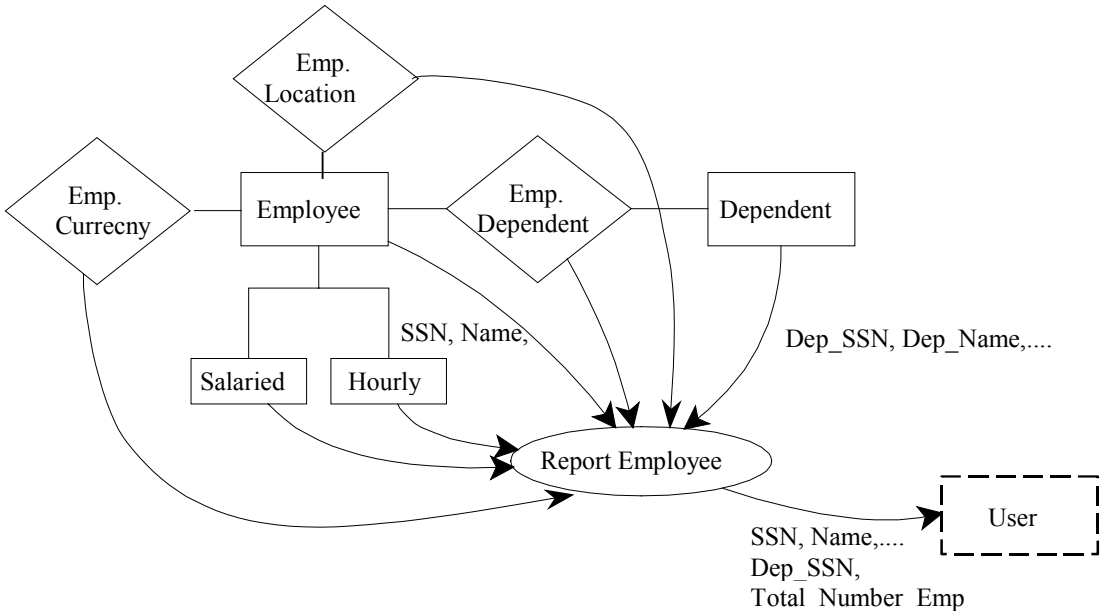

Figure 11: Report Employee process.

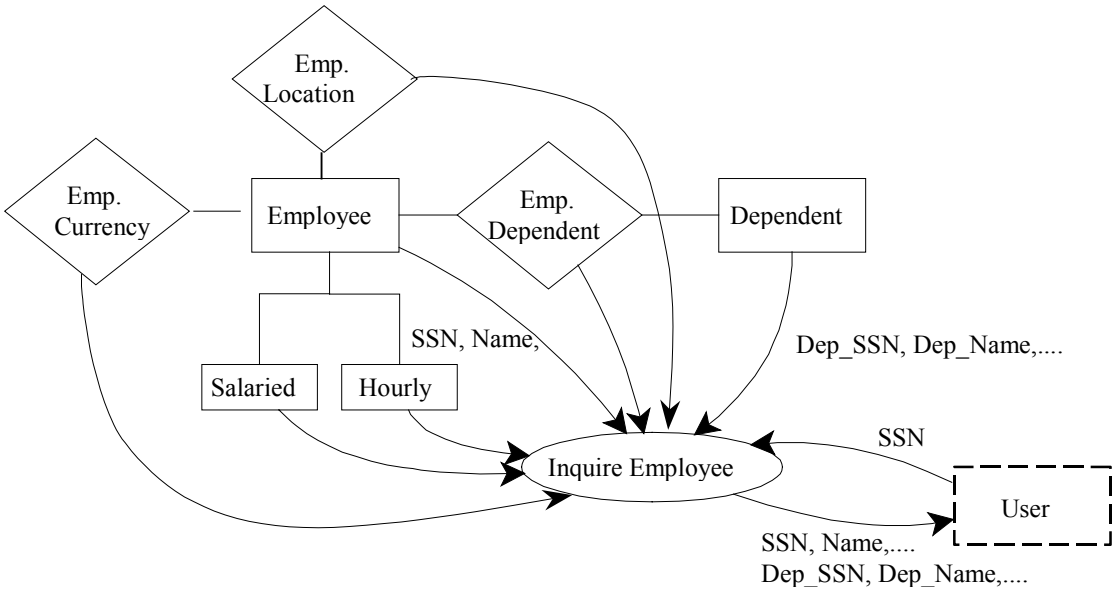

Figure 12: Inquire Employee process.

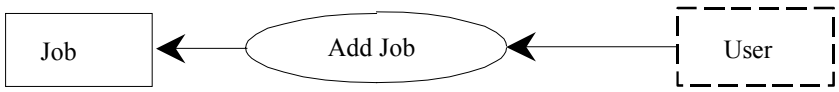

Figure 13: Add Job process.

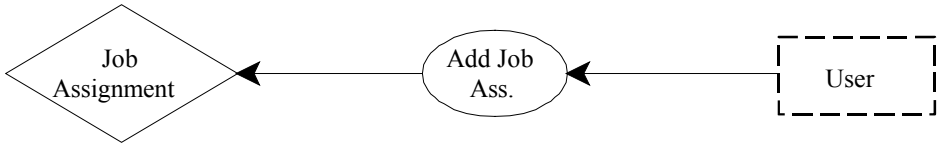

Figure 14: Add Job Assignment process.

After having loaded from the Sicstus interpreter both the system code and the application description, the count is started by calling the goal:

```
| ?- totalFP(hr,FP).
```

where `hr` is the name given to the Human Resources application. The system first computes the total number of FP for the functions of the same type and then sums the totals to find the FP

number for the application. The output (see Figure 15) has the form of a table where each identified data and transaction function is listed together with its number of Function Points, DETs, RETs and complexity. The execution time was 8.85 seconds on a Pentium III at 1.133 GHz with 512 Mb of memory running Microsoft Windows 2000.

The set containing the entities Employee and Dependents, their relationship, Employee's sub-entities Salaried Employee and Hourly Employee, and the relationships Emp. Location and Emp. Currency, is identified by the system as an ILF because it is inside the application boundary and the process Add Employee has data flows to all of them. As regards the complexity count, the ILF has 11 DETs, one for each attribute of its entities, plus 2 DETs for the external keys: one for Location_Name and one for Currency_Location. The system counts 2 RETs for Employee and its two sub-entities, since the hierarchy is total and exclusive, 1 RET for Dependent and no RETs for the relationships Employee Dependent, Emp. Location and Emp. Currency because they do not have any attribute.

The set {Job} is identified as an ILF: it is inside the application boundary and the process Add Job has data flows to the entity Job. {Job Assignment} is an ILF because it is inside the application boundary and the process Add Job Assignment maintains it.

```
| ?- totalFP(hr,F).
Function Point count for application hr

ILF                          FP  DET  RET    Comp
[employee,
salaried_emp,
hourly_emp,
empl_dep,
dependent,
empl_loc,
empl_currency]                7   13    3     low
[job]                         7    4    1     low
[job_assignment]              7    7    1     low

ILF Total                    21

EIF                          FP  DET  RET    Comp
[conversion_rate]             5    4    1     low
[location]                    5    6    1     low

EIF Total                    10

EI                           FP  DET  FTR    Comp

add_emp                       6   13    3    high
change_emp                    6   13    3    high
add_job                       3    4    1     low
change_job                    3    4    1     low
add_job_assignment            6    5    3    high
change_job_assignment         6    5    3    high
delete_emp                    3    1    2     low
delete_job                    3    1    2     low
delete_job_assignment         3    2    1     low
EI Total                     39

EQ                           FP  DET  FTR    Comp

inquire_emp                   3   13    1     low
inquire_job                   3    4    1     low
inquire_job_assignment        3    5    1     low
inquire_location              4    7    2 average
EQ Total                     13

EO                           FP  DET  FTR    Comp
report_emp                    4   14    1     low
report_job                    4    5    1     low
report_job_assignment         4    6    1     low

EO Total                     12


Overall Total                95
Execution time 8.852000 seconds.
F = 95 ?
```

Figure 15: Output for the application `hr`.

The ILF {Job} has 1 RETs and 4 DETs, one for each attribute of Job. The ILF {Job Assignment} has 1 RET and 7 DETs: five DETs for the five attributes of the relationship and 2

DETs for the external keys Social_Security_Number (link to Employee) and Job_Number (link to Job).

As regards EIF identification, we are supposed to be given the information that {Location} is an ILF for the application Fixed Assets and {Currency} is an ILF for the application Currency, since we do not know the processes for these applications. Both {Location} and {Currency} are identified by the system as EIFs because they are ILF for external applications.

Let us now show how transactional functions are identified. Processes Add Employee[2], Add Job Assignment and Add Job are EIs because there are data flows to them from the outside and they maintain an ILF.

The process Report Employee is an EO because there is a data flow to it from an ILF, a data flow F_1 from it to the outside, and F_1 contains one field, Total_Number_Emp, that is not in the flows from the ILF.

Process Inquire Employee is an EQ because there is a data flow to it from the outside, a data flow F_2 from it to the outside and a data flow F_{ILF} to it from an ILF. Moreover, fields in the data flow F_2 are the same as those in F_{ILF}.

Figure 15 shows the results of the complete count, including processes here not considered in details. The total number of FP is the same as the one obtained in [18].

---

[2] Note that we have counted 13 DETs for Add Employee and Change Employee instead of 12 as reported in [18] because in order to compute the value of the field US_Hourly_Rate of Hourly the application must retrieve the Conversion_Rate_To_Base_Currency from the Currency application. Therefore, the conversion rate crosses the boundary of the application and should be counted as a DET.

We have applied the system also to the five applications described in [20]. They are relative to the same domain of warehouse management and they are partly overlapping, in the sense that they share part of the data and part of the processes. For this reason they are defined in [20] as an application *portfolio*. In practice they can be considered as a number of application personalized for different customers that a software company has developed regarding a single domain.

The five applications are named W, M, C, LC and LS. FUN was applied to these systems and the measurements obtained were the same as those reported in [20]. Table 5 shows these values.

Application	FP
W	77
M	40
C	49
LC	56
LS	31

Table 5: numbers of FP for the applications in [20].

Finally, an application of significant size has been measured. The application has been specified by one of the authors [21] and manages the information of a company selling sport apparel. The ER diagram contains 15 entities and 19 relationships, while the DFD contains 50 processes. The result obtained with FUN has been compared with the one of an independent certified human FP counter: Fun has obtained 254 FP while the human counter has obtained 257 FP. The difference is due to the fact that the human counter has counted one DET more than FUN for every transaction function. This extra DET is considered for taking into account the

command that the user gives to perform the function. However, in IFPUG case study [18] such DETs are not counted therefore we feel FUN provides an answer more adherent to the IFPUG guidelines.

7 Related Work

A number of tools for the measurement of FP have been developed. We distinguish tools developed by academic researchers from tools developed by commercial companies.

As regards tools developed in academia, the authors in [11] present a system for the automated measurement of FP from the design specification of an application expressed using UML. In particular, the system starts from sequence diagrams and class diagrams: sequence diagrams shows examples of interaction between the applications and the user or other applications while class diagram described the structure of the object-oriented design of the application. When an UML diagram is available, then the tool described in [11] is more suitable than our tool. The opposite is true in the case of a specification expressed as a combination of an ER with a DFD.

As regards commercial tools, IFPUG has established a certification program for commercial FP counting tools that recognizes three types of tools: type 1, 2 and 3 [3]. In type 1 tools the user performs the identification of the functions and the tool is used only for calculations. In type 2 tools the count is determined in an interactive way: the systems asks questions to the user and determines the count on the basis of the answers. In type 3 tools the system performs the count autonomously on the basis of a stored description of the application. At the moment there are three recognized type 1 tools, only one type 2 tool, namely PQMPlus by Q/P Management [22], and no type 3 tool. We believe that our system is to be considered as a type 3 tool, being completely autonomous in the calculation of the count from the ER-DFD diagram.

Among the commercial tools, however, there exist two tools that claim that they are able to automatically count FP from DFD or ER-DFD: Saver [23] and Xupper & Xradian [24]. However, from the information available on the web it was not possible to verify if the counting is done in a fully automatic way and which are the assumptions they make. Moreover, they are targeted exclusively to the Japanese market, being available only in Japanese.

Our analysis, besides leading to a counting tool, has the benefit of increasing our understanding of the FP counting rules when counting from ER-DFD, removing the ambiguities that are present in the counting rules and translating them into simple rigorous rules by making only a limited number of assumptions.

8 Conclusions and Future Work

In this paper we have presented a system for the automatic counting of the Function Point metric starting from an Entity Relationship - Data Flow Diagram, a formalism that integrates ER and DFD by replacing the data stores of DFD with the entities and relationships of ER. The FP counting rules have been specialized for the case of ER-DFD and made rigorous by making a number of simplifying assumptions. The rigorous rules express simple properties of the ER-DFD graph that could be easily checked by means of a Prolog program.

The paper makes thus a double contribution: on one side, it provides a translation of the IFPUG counting rules into a rigorous form that can help reduce the discrepancy between different counters of the same application; on the other side it provides a system for the automated measurement of Function Points.

In the future, we intend to apply FUN to more cases and compare the results with those obtained by human experts. Moreover, work will be devoted to the integration of the system with

tools that generate the description of the system in FUN notation starting from a drawing made by the use on the screen. To this purpose, the suite of graph drawing tools Graphviz [25] can be used and, in particular, the tool dotty that allows the user to draw a graph on the screen and, from it, to generate a textual description of the graph in the dot language. It would be then sufficient to write a translator from the dot language to the FUN language to have a tool that counts FP directly from a drawing of the ER-DFD. Finally, by writing a translator from the FUN notation to the dot notation, it would be possible to generate a drawing of the ER-DFD given a description in FUN notation.

9 Acknolwedgements

The authors would like to thank Roberto Meli and Franco Perna for having performed the count of the application in [21].

References

[1] Albrecht, A. (1979) Measuring application development productivity. *Proc. Joint SHARE/GUIDE/IBM Applications Development Symposium*, Monterey, CA, 14-17 October, pp. 83-92, IBM, Armonk, NY .

[2] Albrecht, A. and Gaffney J. (1983) Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans. Software Eng.*, **9(6)**, 639-648.

[3] International Function Points User Group (2000) *Function Point Counting Practices Manual*, *Version 4.1.1*, International Function Points User Group, Princeton Junction, NJ.

[4] International Function Points User Group (2002) `http://www.ifpug.org/`, accessed 28 November 2002.

[5] International Function Points User Group (2002) `http://www.ifpug.org/bimonthly/iso1102.htm`, accessed 28 November 2002.

[6] Furey, S. (1997) Why We Should Use Function Points. *IEEE Software*, **14(2)**, 28-30.

[7] Kitchenham, B.A. (1997) The Problem with Function Points. *IEEE Software*, **14(2)**, 29-31.

[8] Kemerer, C. (1993) Reliability of Function Point Measurement: A Field Experiment. *Communications of the ACM*, **36(2)**, 85-97.

[9] Low, G. C. and Jeffery, D. R. (1990) Function Points in the Estimation and Evaluation of the Software Process. *IEEE Transaction on Software Engineering*, **16(1)**, 64-71.

[10] Lamma, E., Mello, P. and Riguzzi, F. (1998) A System for Measuring Function Points. *Proceedings of the Sixth Intl. Conference on Practical Applications of Prolog and Forth Intl. Conference on Practical Applications of Constraint Technology (PAPPACT98)*, London, March, pp. 41-60, The Practical Application Company Ltd , London.

[11] Uemura, T., Kusumoto, S. and Inoue, K. (2001) Function-point analysis using design specifications based on the Unified Modelling Language. *Journal Of Software Maintenance And Evolution: Research And Practice*, **13**, 223–243.

[12] Swedish Institute of Computer Science (1995) *SICStus Prolog User Manual, Release 3#0*, Swedish Institute of Computer Science, Kista, Sweden.

[13] Chen, P. P. (1976) The Entity-Relationship model. Toward a unified view of data. *ACM Transactions On Database System*, **1(1)**, 9-36.

[14] DeMarco, T. (1978) *Structured Analysis and System Specification.* Yourdon Press, New York.

[15] Fuggetta, A., Ghezzi, C., Mandrioli, D. and Morzenti, A. (1988) VLP: a Visual Language for Prototyping. *IEEE Workshop on Languages for Automation: Symbiotic and Intelligent Robots*, College Park, MD, 29-31 August, pp. 134-149, IEEE, New York, NY.

[16] Bratko, I. (1986) *Prolog Programming for Artificial Intelligence.* Addison Wesley, Boston, MA.

[17] International Function Points User Group (2001) *Function Point Counting Practices: Case Studies, Case Study 1, Release 2.0*, International Function Points User Group, Princeton Junction, NJ.

[18] International Function Points User Group (2001) *Function Point Counting Practices: Case Studies, Case Study 2, Release 2.0*, International Function Points User Group, Princeton Junction, NJ.

[19] International Function Points User Group (2001) *Function Point Counting Practices: Case Studies, Case Study 3, Release 2.0*, International Function Points User Group, Princeton Junction, NJ.

[20] Fetcke, T. (1999) The Warehouse Software Portfolio: A Case Study in Functional Size Measurement, *Report No. 1999-20*, Technische Universität Berlin, Fachbereich Informatik, Berlin, `http://user.cs.tu-berlin.de/~fetcke/papers/Fetcke1999b.pdf`, ISSN 1436-9915.

[21] Riguzzi, F. (2003) Specification of the Application SuperSport with ER-DFD, *Report No. CS-2003-01*, Dipartimento di Ingegneria, Università di Ferrara, `http://www.ing.unife.it/aree_ricerca/informazione/cs/technical_reports/CS-2003-01.pdf`.

[22] Q/P Management (2002) *PQMPlus*, http://www.qpmg.com/pqmplus.htm, accessed 27 December 2002.

[23] Vest Software Co. (2003) *SAVER*. http://www.vest.co.jp/Saver/saver.php3,

[24] Ken System Development Co. (2003) *Xupper & Xradian*. http://www.kensc.co.jp, accessed 7 July 2003.

[25] AT&T (2003) *Graphviz*. http://www.research.att.com/sw/tools/graphviz, accessed 7 July 2003.