Intelligent Agent based Mapping of Software Requirement Specification to Design Model

Emdad Khan
College of Computer and Information Sciences
Al-Imam Muhammad Ibn Saud Islamic University
Riyadh, Saudi Arabia
Email: emdad {at] ccis.imamu.edu.sa

Mohammed Alawairdhi
College of Computer and Information Sciences
Al-Imam Muhammad Ibn Saud Islamic University
Riyadh, Saudi Arabia

Abstract— ***Automatically mapping a requirement specification to design model in Software Engineering is an open complex problem. Existing methods use a complex manual process that use the knowledge from the requirement specification/modeling and the design, and try to find a good match between them. The key task done by designers is to convert a natural language based requirement specification (or corresponding UML based representation) into a predominantly computer language based design model – thus the process is very complex as there is a very large gap between our natural language and computer language. Moreover, this is not just a simple language conversion, rather a complex knowledge conversion that can lead to meaningful design implementation.***

In this paper, we describe an automated method to map Requirement Model to Design Model and thus automate / partially automate the Structured Design (SD) process. We believe, this is the first logical step in mapping a more complex requirement specification to design model. We call it IRTDM (Intelligent Agent based requirement model to design model mapping). The main theme of IRTDM is to use some AI (Artificial Intelligence) based algorithms, semantic representation using Ontology or Predicate Logic, design structures using some well known design framework and Machine Learning algorithms for learning over time. Semantics help convert natural language based requirement specification (and associated UML representation) into high level design model followed by mapping to design structures. AI method can also be used to convert high level design structures into lower level design which then can be refined further by some manual and/or semi automated process. We emphasize that automation is one of the key ways to minimize the software cost, and is very important for all, especially, for the "Design for the Bottom 90% People" or BOP (Base of the Pyramid People).

Keywords- ***Software Engineering, Artificial Intelligence, Ontology, Intelligent Agent, Requirements Specification, Requirements Modeling, Design Modeling, Semantics, Natural Language Understanding, Machine Learning, Universal Modeling Language (UML), ICT (Information and Communication Technology and BOP (Base of the Pyramid People).***

I. INTRODUCTION

Converting requirement specification or model to design model followed by an implementation is an important part of software engineering, especially for a large scale software. It is both information conversion and knowledge conversion, and it involves both art & science. Hence the process is complex. In fact, the various levels of abstractions involved in such mapping (e.g. from requirement model to design model, to architecture, to implementation) make the process even more complex. Designers use their expertise and various available tools to successfully complete the process. Since software cost is an important factor for many organizations (in fact, it is a key factor for almost all countries as it is a significant part of GDP, Gross Domestic Products), it is important that we keep the software cost minimal. This is even more true for underdeveloped and developing countries dominated by BOP (Base of the Pyramid People) -many of them are poor i.e. income is less than $2 per day. Minimizing software cost will help such countries to afford ICT (Information and Communication Technologies) and associated software; and thus will provide the benefits of the Information Age to such population. This fits well, with "Design for the bottom 90% people". Automation is one of the key ways to minimize the software cost [11].

Many researchers have been working on automating various parts of the software engineering including software development process. E.g. to help architectural design, various models have been proposed like Structural Models, Framework Models, Dynamic Models, Process Models and Functional Models ([2], [3]). A number of different Architectural Description Languages (ADLs) have been developed to represent these models ([4], [6]). Similarly, to help requirement modeling, various languages have been developed e.g. Requirement Modeling Language, RML ([1], [7], [10]). However, we could not find any citation regarding automatically mapping a Requirement Model to a Design Model. A few somewhat related researches are covered in ([13], [15]).

In this paper, we present an Intelligent Agent (IA) based automated method to map Requirement Model to a Design Model. It is called **IRTDM** (Intelligent Agent based requirement model to design model mapping). The IA uses Artificial Intelligence (AI), semantic representation using Ontology or

Predicate Logic, Design Structures (DS) using some well known design framework and Machine Learning algorithms for learning over time. We specifically focus on mapping Requirement Model to Architecture. Mapping to other key software areas / steps (e.g. converting the architecture into operational software) are also possible using similar approach but not covered in this paper.

Section II provides a brief high level overview of IRTDM (Intelligent Agent based requirement model to design model mapping). Section III describes the basics of the Flow-Oriented Requirement modeling to Data-Flow architecture mapping method as done by experienced designers. Section IV describes an automated version of Section III using Natural Language Processing / Understanding, Artificial Intelligence and an Intelligent Agent. Section V describes the Architecture and Algorithms for more general and versatile Intelligent Agent. It also briefly discusses how to apply the concept for other types of mapping, Section VI describes future works and Section VII provides conclusions.

II. HIGH LEVEL OVERVIEW OF IRTDM

There is a good correspondence between requirement model and design model (Fig.1). Various parts of the Requirement Model have corresponding mapped parts in the design model. E.g. class-based elements map to data / class, architecture and component design parts in the design model. In fact, designers use such basic mapping as a basis to come up with an architecture. Designers also use various levels of architectural abstractions (e.g. Architectural Genre, Architectural Styles, Archetypes) to come up with the structure showing key blocks or components. Our main theme is to use designers approach to come up with an automated approach. It is important to note that for some cases there is no practical mapping from requirement model to some architectural styles. But for many cases such mapping exists. A good example is mapping Flow-Oriented Requirement modeling to Data-Flow architecture style. Since enough abstractions already exist and the manual method is understood reasonably well, we can convert the same into appropriate steps that can be done by an Intelligent Agent (IA) i.e. IA in IRTDM. First we discuss a simple IA to automatically handle Flow-Oriented Requirement modeling to Data-Flow architecture. Then we discuss more general IA.

The key issues a general IA needs to address are:

1. Use of proper rules in doing the mapping.
2. Use of semantics to ensure correct mapping.
3. Use of appropriate rules and semantics to help map / transform one architectural style to another (e.g. Data-flow architecture to Layered architecture).
4. Use of Learning to improve the outcome.
5. Use of Verification to ensure correctness.
6. Help Ensure that Implementation (coding) can also be automated in a similar way.
7. Other key issues as appropriate (e.g. refactoring, generating test vectors and performing basic tests).

III. FLOW-ORIENTED REQUIREMENT MODELING TO DATA-FLOW ARCHITECTURE MAPPING

A mapping technique called Structured Design (SD) is often characterized as a data flow-oriented design method [10] as it provides a convenient transition from a data flow diagram (DFD) to software architecture. Such transformation involves the following 6 steps:

a. The type of data (information) flow is established
b. Flow boundaries are determined
c. The DFD is mapped into the program structure
d. Control hierarchy is defined
e. Resultant structure is refined using design measures and heuristics, and
f. The architectural description is refined and elaborated.

In order to design optimal module structure and interfaces two principles are crucial [10]:

- *Cohesion* which is "concerned with the grouping of functionally related processes into a particular module" and

- *Coupling* relates to "the flow of information, or parameters, passed between modules. Optimal coupling reduces the interfaces of modules, and the resulting complexity of the software".

[**Note:** In general, Structured Design (SD) and Structured Analysis (SA) are methods for analyzing and converting business requirements into specifications and ultimately, computer programs, hardware configurations and related manual procedures. SA includes Context Diagram, Data Dictionary, DFD, Structure Chart, Structured Design and Structured Query Language (SQL)]

One form of information mapping is called **Transform mapping** where incoming data is transformed into an internal form by a transform center. The transformed data then flows to external world using outgoing flow. Another form of information mapping is called **Transaction mapping** in which a single data item triggers one or a number of information flows that effect a function implied by the triggering data item. The data item is called a transaction.

The above mentioned steps are done by designers (all types of designers including database and data warehouse designers and system architects) using the Requirement Model (in this case the Flow-oriented model) and the design

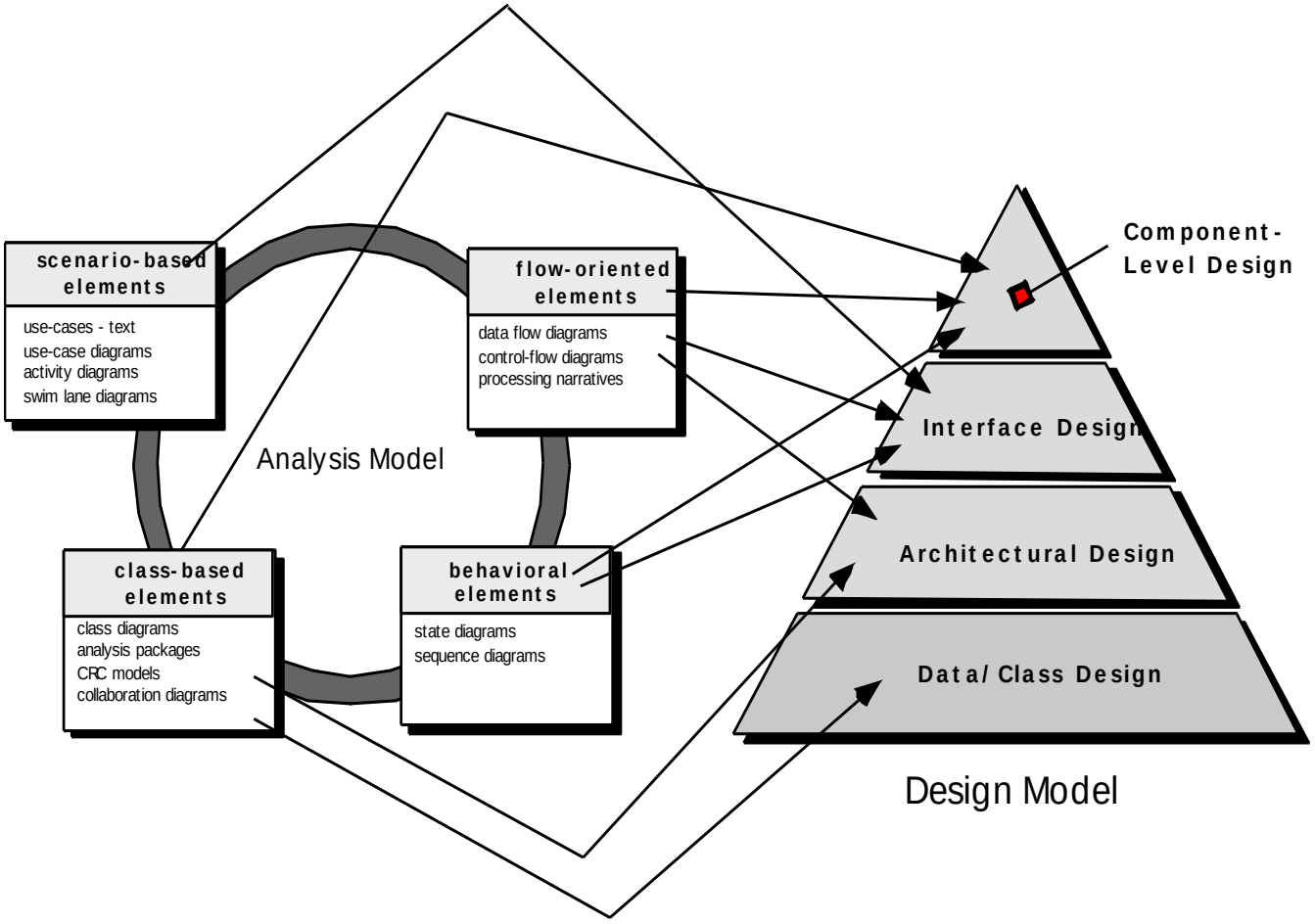

Fig. 1 Flow-Oriented Requirement Modeling to Data-Flow Architecture Mapping (Courtesy [10]).

structures including Design Genre, Design Styles (in this case data flow architecture), set of archetypes (e.g. Controller, Detector, Indicators, Node), basic classes (some of which are described in the Requirement Model) and some basic design guidelines. Refer to "Software Engineering: A Practitioner's Approach" by Roger Pressman [10] for a detailed example. We basically automate these steps using NLU, AI and an Intelligent Agent as described below in Sections 4 and 5.

IV. AUTOMATING FLOW-ORIENTED REQUIREMENT MODELING TO DATA-FLOW ARCHITECTURE MAPPING

Converting Flow-Oriented Requirement Modeling to Data-Flow Architecture is a good start because of its simplicity. In this case there is a direct correspondence between the requirement modeling steps and architectural mapping steps as both use the same DFD.

4.1 *Basic Ideas*

Use the requirement modeling flow information and match it using AI rules to the corresponding Data-Flow Architecture. Since there is 1-1 correspondence (refer to Fig. 1), Flow-Oriented elements have 1-1 correspondence with the Design

Model blocks like Architectural Design), developing such rules are straight forward (refer to Sections 4.2, 4.3 and the example in Section 5). The rules are needed mainly to map DFD to the program structure, determine control hierarchy, complete refinement and elaboration.

Referring to Fig. 1, there is a 1-1 correspondence from the DFD Requirement Model to Architectural Design,

Interface Design and Component Level Design. Thus, we need appropriate rules to map to all such design levels. Cohesion and coupling are appropriately used to ensure optimal design module structures and interfaces. Any standard automatic / semi-automatic technique can be used to determine the optimal design module structures and interfaces. All these key steps can be iterated during the refinement process (steps #e and #f in Section 3).

4.2 *Requirement Modeling and Natural Language Processing (NLP)*

Requirement Modeling methods usually use natural language words or equivalent methods. For example, in a Use Case diagram, the concept is expressed using natural language type concept. Class based, Behavioral based and DFD approaches also use natural language type concept. Thus, it is important to use Natural Language semantics and Natural Language Processing (NLP) in automating the mapping of Requirement Modeling to Design process. In case of DFD based modeling (as already mentioned), we would need semantics and NLP to map DFD to the program structure, determine control hierarchy, complete refinement and elaboration.

Besides, in a typical design,

a. The software must be placed into context i.e. the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction.
b. A set of architectural archetypes should be identified - an *archetype* is an abstraction (similar to a class) that represents one element of system behavior.
c. The designer specifies the structure of the system by defining and refining software components that implement each archetype.

NLP becomes handy in automating all these activities. Let's use an example to demonstrate the use of semantics and NLP:

Refer to Fig.2 – it shows a simple DFD with reasonable details (i.e. say level 3 DFD). An analog signal is input to an Analog to Digital conversion unit (the Transform center circle or bubble #2) after doing some filtering operation by circle #1. The transform center outputs the Digital signal in two format – binary (bubble #3) and hexadecimal (bubble #4). All bubbles are labeled with words that are easily understandable to human being as these are natural language words. Our goal is to use the semantic meaning of these words to come up with a design structure as designers usually do.

Consider the words "Analog to Digital Conversion" in bubble #2. The semantic meaning of this is "Conversion from an analog signal to digital signal takes place here" (see Section 4.3 below how such semantics is derived / programmed). Once the program knows this semantics, it can determine the corresponding design archetypes and top level design box using AI rules which are based on the domain knowledge, semantics, and the DFD itself. Fig. 3 shows the corresponding design structure. Such as structure is achieved using the following concept (the corresponding rules are given in Section 4.3):

1. The boundaries shown in Fig.2 are used to focus on the design of bubble #2. This is as per standard DFD based design process as outlined in Section 3.
2. Such boundaries can easily be done by representing the DFD using a Graph which can be implemented using netlist.
3. Since bubble #2 is taking one input and producing 2 outputs of different data formats, bubble #2 is doing a "Transform flow".
4. The outputs of the transform flow are detailed out in the DFD itself. So, corresponding design blocks can easily be constructed (Fig. 3 shows this using **DFD based mapping to a Call and Return architecture**).
5. As bubble #2 is doing a transform operation, it needs to do a "control function" in addition to do the main "transform function". This is again part of the standard design process that designers use in a Structured Design.
6. Netlist of the DFD is used to move and identify the new boundaries (by the automation software i.e. IA), find the new transform center and complete the design for new transform center, e.g. Binary Format-3 bubble and Hex Format bubble (Fig. 3).

The following Section implements these concepts using semantics, NLP and AI. And all these are part of the Intelligent Agent, IA.

4.3 *Predicate Calculus and Mapping Rules*

The rules mentioned above can be represented by Predicate Calculus rules. Predicate Calculus can also be used to define semantics. We can also use Ontology to define semantics. In this paper, we are using Predicate Calculus to describe the rules and semantics.

Consider the words "Analog to Digital Conversion" in bubble #2 in Fig. 2 (as described in Section 4.2). The semantic meaning of this is "Conversion from an analog signal to digital signal takes place here" or simply "Conversion from an analog signal to digital". In predicate calculus (or First order logic, FOL), we can use the following to represent this semantics:

Converts (Convert to Digital -2, AnalogToDigital) ……. (1)

AnalogToDigitalConverter (Convert to Digital -2) …..…(2)

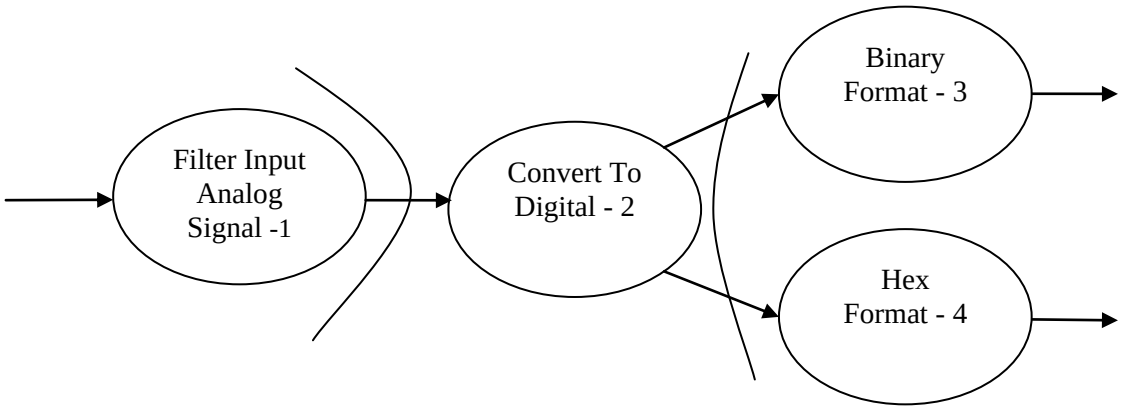

Fig. 2: A Simple Transform Flow DFD. "Convert to Digital" circle (bubble) is the Transform Center. Input is an Analog signal which is converted by the Transform Center into Digital signal with two formats – Binary and Hexadecimal. The semantics of the "label" words of each bubble are used to automate the Design Process – see texts in Section 4.2 for details.

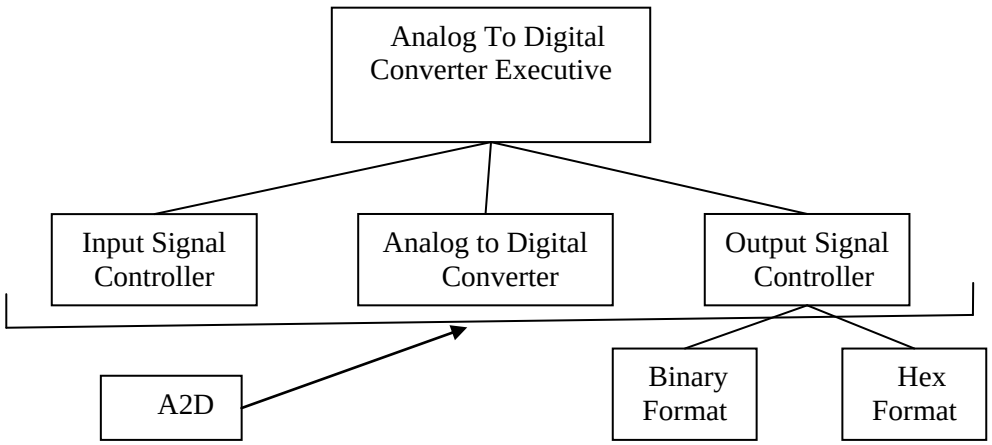

Fig. 3: Design structure constructed by using the DFD in Fig. 2. Semantics of the bubbles 2, 3 and 4 in Fig. 2 and corresponding rules are used to make the construction. Semantics and all associated rules are implemented using First Order Logic (FOL). See Section 4.2 and Section 4.3 for details.

Converts (AnalogToDigitalConverter, AnalogToDigital) ……..….(2a)

When "Convert to Digital -2" label is seen in DFD bubble #2, the semantics determines that this is an analog to digital converter. Hence, all the design structures have the key blocks needed to implement the function of an analog to digital converter (Fig. 3).

To make it more general, we use universal quantifier "for all" i.e. ∀ to say,

"All analog to digital converters convert analog signal to digital signal" …….. (3a)

Which can be written in FOL

∀x AnalogtoDigitalConverter (x) ⇒ Converts (x, AnalogToDigital) ……..…… …….(3b)

Using the universal quantifier, we allow to use any analog to digital converter in our knowledgebase or library.

[Note: mathematically, x can be any variable, including an instance of a non-AnalogToDigitalConverter [8]. This, however, can be avoided in various ways. We take care of this by only allowing analog to digital converters in the corresponding library]

In addition, an Executive control block (Analog To Digital Converter Executive) and a few other associated control blocks (e.g. input signal controller and output signal controller) are generated (Fig. 3) as per standard design technique used in DFD model. Similarly, using the semantics of other bubbles, blocks to handle the binary and hex format are constructed. The FOL rules are used to describe all these as shown below:

If x is AnalogToDigitalConverter then Blocks are
"Analog To Digital Converter Executive"
AND "Analog To Digital Converter"
AND "Input Signal Controller"
AND "Output Signal Controller" ……...…..(4)

If x is Binary Format then Blocks are "Binary Format" …… …….(5)

If x is Hex Format then Blocks are "Hex Format" ……...…...(6)

The actual blocks for the analog to digital converter can have more than one block and also multi-level blocks as appropriate. But the whole thing can be labeled in the knowledge base as one block (e.g. A2D as shown in Fig. 3) so that it is placed properly when such a rule (i.e. equation 4) is fired (see Section 4.3 for more details). The same is true for all other blocks and associated rules (e.g. Binary and Hex format blocks in Fig. 3). Note, in a rule (e.g. equation 4), the semantics that it is an AnalogToDigitalConverter is derived using equations (1) and (2) [see Section 4.4 for more details].

It may seem trivial that we could just use the label directly to construct the design structure using appropriate blocks. Yes, it is true for simple cases. But label may be more complex (can have more words and mean multiple operations), the format and words may vary considerably and the like. Use of NLP & FOL can define the meaning in a more flexible and reliable way, especially for complex cases. NLP & FOL become more important for refining the resultant structure (step #e in Section 2), and when the architectural description is refined and elaborated (step #f in Section 2). See Section 5 and Section 6 for more details.

4.4 Design Structures

In order to properly execute steps (#c to #f) in Section 2, namely,

c.The DFD is mapped into the program structure
d. Control hierarchy is defined
e. Resultant structure is refined using design measures and heuristics, and
f. The architectural description is refined and elaborated,

designers follow various policies and processes. An architectural genre (e.g. Operating System or Artificial Intelligence), architectural style (e.g. Data-centric or Call and Return) and a set of Archetypes (e.g. Nodes, Detector, Indicator, Controller) need to be selected / defined. These are heavily influenced by designer's experience and knowledge. Such knowledge and experience need to be put in the knowledgebase using appropriate rules and predefined structures and blocks. Here, the designers have the option to make the automated system very efficient. Such structures and blocks need to be refined on a regular basis for continuous improvement.

To make the design modeling & construction of the design structure flexible and efficient, and to better support refinement and elaborations, design structures / blocks needs to be configurable via some parameters. This scheme will better support the flexibility in the A2D implementation as mentioned in Section 4.3.

4.5 The Automation Process

The automation process involves the following key steps:

1. Create a good knowledgebase (KB) that has key information that designers follow in converting a

requirement model to design model or structure. Designers use various policies and processes. Such a knowledgebase need to include all architectural genre, architectural styles, and set of archetypes.

2. The KB also would need to include all rules to convert a DFD (other representations used for Requirement Modeling) to design structures and blocks.
3. Design library needs to have all the key structures, blocks, components with appropriate parameterization.
4. Establish mechanism to continuously improve the library and the design process based on learning from previous design structures. This part can be automated using separate rules and semantics.

Once the above keys steps are completed, the IA (see Section 5), can take a DFD directly and produce a design structure as shown in Fig. 3. IA accomplishes this by taking the DFD netlist and implementing (i.e. converting) each bubbles using the semantics of the bubbles and the rules.
The facts and the rules are combined using an inference mechanism, like **Modus-Ponens**.

Multiple rules can be fired and **Forward Chaining, or Backward Chaining** can be used to derive the final design structure. A short example is shown below using the AnalogToDigitalConverter example discussed in Sections 4.2 and 4.3:

AnalogToDigitalConverter (Convert to Digital -2) ……..(2)
[a Fact - Convert to Digital -2 is an AnalogToDigitalConverter]

∀x AnalogtoDigitalConverter (x) ⇒ Converts (x, AnalogToDigital) ……..(3b)
[Rule – for all x, if x is an AnalogtoDigitalConverter, then it converts AnalogToDigital]

[Using Modus-Ponen] Converts(Convert to Digital -2, AnalogToDigital) [**New derived fact**]

Note that the new derived fact by using Modus-Ponens is already shown in equation (1). But it is shown there to express the semantics of the bubble #2 in Fig. 2. But it is not used to represent a fact there. When it is derived as a fact, then equation (4) will fire and will create the design structure (Rule represented by equation (4) is not an implication as used in equation 3(b). However, it can be converted to an implication form). Also, while Forward and Backward Chaining are **sound**, neither is **complete**. This means that there are valid inferences that cannot be found using these methods alone. An alternative inference technique called **Resolution** is sound and complete but computationally expensive [8].

V. INTELLIGENT AGENT

An Intelligent Agent, IA implements the automation described in Section 4.5. It also performs other functions including some advanced functions needed to handle requirement models other than DFD i.e. Class based, Use Case based and State based models or their combinations that may include DFD. The key functions of IA are mentioned in Section 2. The implementation of key functions are described in Sections 3 & 4 for **DFD based mapping to a Call and Return architecture.** Such implementations are, in general, applicable for all other mappings with some refinements. Fig. 4 shows the architecture of a general IA. A few key functions not yet described are:

1. Use of appropriate rules and semantics to help map / transform one architectural style to another (e.g. Data-flow architecture to Layered architecture).

2. Use of Learning to improve the outcome.

3. Use of Verification to ensure correctness.

Architectures for which direct mapping does not exists, the mapping process becomes complex. The designers approach the translation of requirements to design for such cases using their knowledge, more analyses and considering more architectural tradeoffs. Although there is no simple steps like steps #a to steps #f as mentioned in Section 2 for DFD based mapping, the designer's approach can be captured into similar flow and steps but with more natural language descriptions. Thus, for such cases, the issue of using NLP becomes more important and semantics & rules become more complex.

The learning over time can be implemented using any standard good learning algorithms. The verification process can be implemented by allowing to perform some basic tests on the constructed system. Each component will have netlist or behavioral model representation which can take input vectors and verify the outputs with some predefined expected outputs (in compliance with the specification). In some cases, formal verification can be done using formal mathematical specification of the software.

VI. FUTURE WORKS

The semantics represented by FOL and other similar techniques are good but they work satisfactorily mainly for small domain. As shown in Section 4.3, we need to define semantics for almost everything i.e. existing schemes do not allow to automatically derive new semantics from semantics of existing words. In ([14], [16]) we have mentioned that while traditional approaches to Natural Language

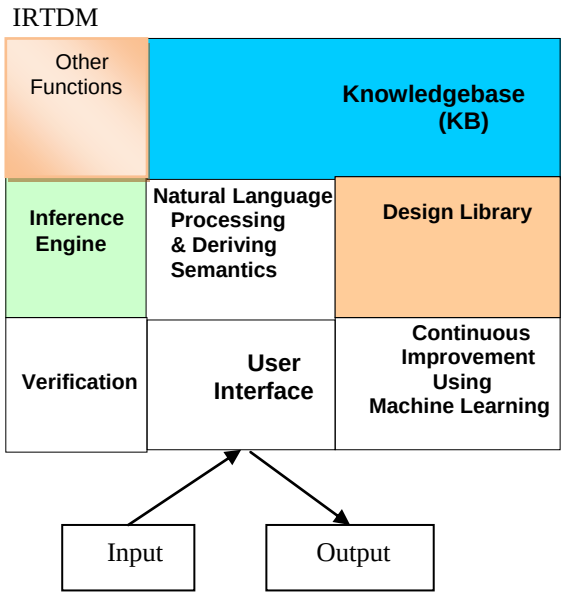

Fig. 4: IRTDM - Intelligent Agent for requirement model to design model mapping. Shows all the key blocks. The KB and Design Library can reside outside. Input is mainly the requirement model and output is mainly the design structure and blocks.

Understanding (NLU) have been applied over the past 50 years and have had some good successes mainly in a small domain, results show insignificant advancement, in general, and NLU remains a complex open problem. NLU complexity is mainly related to **semantics**: abstraction, representation, real meaning, and computational complexity. We argued that while existing approaches are great in solving some specific problems, they do not seem to address key Natural Language problems in a practical and natural way. In [12], we proposed a Semantic Engine using **Brain-Like approach (SEBLA)** that uses Brain-Like algorithms to solve the key NLU problem (i.e. the semantic problem) as well as its sub-problems.

SEBLA can calculate semantics of sentences using the semantics of words and the semantics of a paragraph using the semantics of the sentences. Enhanced semantics capability is needed to handle complex mapping cases mentioned in Section 5. We plan to use SEBLA for such cases.

We also plan to use SEBLA to automate / partially automate the implementation of the architecture into final software form (i.e. converting the architecture into operational software). Note that the automation presented in this paper is not the implementation in final software form; it is rather automating the mapping to design structure or architecture or blueprint of the desired system.

VII. CONCLUSIONS

IRTDM (Intelligent Agent based requirement model to design model mapping) will significantly help today's large software development process. It takes long time to manually map the requirement model to a design model. As the software size gets bigger and bigger (a common trend in the industry), this process will become much more complex, and need for an automation of this process will become mandatory. In fact, automation is already mandatory to handle existing software design / development if we focus on the design for the bottom 90% people (the so called Base of the pyramid people, BOP).

IRTDM will also increase the reliability and correctness of the said mapping and associated software. Moreover, with Natural Language Processing / Understanding and Artificial Intelligence (AI), the IA (Intelligent Agent) can map the design model to high level design components, thus further providing significant help in already very complex software engineering process.

Thus, our IRTDM will save significant cost for software which is a key component of the total yearly expense of most countries. Lower software cost implies lower price for buying new software; thus allowing many more people in the world to enjoy the benefits of the Information Age.

We have emphasized the need for enhanced Natural Language Processing / Understanding to better handle semantics, especially, for the complex software development cases. Use of natural semantics (e.g. SEBLA [12]) is the key to achieve this which we plan to do next.

REFERENCES

[1] Greenspan, S. et al., "**A** requirements modeling language and its logic, Information Systems, v.11 n.1, p.9-23, 1986.
[2] Abowd, G. et al, " Structural Modeling: An Application Framework and Development Process for Flight Simulators", CMU Technical Report CMU/SEI-93-TR-014, Aug, 1993.
[3] Garlan, D and M. Shaw, "An Introduction to Software Architecture", Advances in Software Engineering and Knowledge Engineering, Vol. I, World Scientific Publishing Company, 1995.
[4] Clements, P., "A Survey of Architectural Description Languages", Paul C. Clements, Software Architecture, Software Engineering Institute, March 1996.
[5] Structured Analysis - http://en.wikipedia.org/wiki/Structured_analysis .
[6] Architecture Analysis and Design Language, Software (AADL), Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA, 2004.
[7] James Rumbaugh et al "The Unified Modeling Language Reference Manual (2nd Edition) July, 2004, | ISBN-10: 032171895X, Addison-Wesley.
[8] Buschmann, F., et al., "Pattern-Oriented Software Architecture, A System of Patterns", Wiley 2007.
[8] Jurafsky, D., et al., "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition", Pearson / Prentice Hall, 2009.
[10] Pressman, R., "Software Engineering: A Practitioner's Approach", McGrawHill, 2010.

[11] Khan, E., "Internet For Everyone: Reshaping the Global Economy by Bridging the Digital Divide" published in Aug, 2011 by iUniverse; 978-1-4620-4251-7 (SC ISBN) 978-1-4620-4250-0 (HC ISBN).

[12] Khan, E., "Natural Language Understanding Using Brain-Like Approach: Word Objects and Word Semantics Based Approaches help Sentence Level Understanding", Applied to U.S. Patent Office, 2012.

[13] Process Model Requirements Gap Analyzer *http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture-Process-Model-Requirements-Gap-Analyzer.pdf Accenture* 2012.

[14] Khan, E., "E. Khan, "Natural Language based Human Computer Interaction: a Necessity for Mobile Devices", INTERNATIONAL JOURNAL of COMPUTERS AND COMMUNICATIONS, (NAUN & UNIVERSITY PRESS) Dec. 2012.

[15] Okud H. Eat al, "Experimental development based on mapping rule between requirements analysis model and web framework specific design model", SpringerPlus Journal 2013, **2**:123 doi:10.1186/2193-1801-2-123.

[16] E. Khan, "Addressing Big Data Problems using Semantics and Natural Language Understanding**"**, 12th WSEAS International Conference on TELECOMMUNICATIONS and INFORMATICS (TELE-INFO '13) in Baltimore, MD, USA, September 17-19, 2013.

.