Transformations for Rapid Prototyping of Time-Critical Applications

Shi-Kuo Chang¹, Zhoulan Zhang¹, Colin J. Ihrig¹
Paolo Maresca² and Valentina Ternelli²
¹Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260 USA
chang@cs.pitt.edu
and
²Department of Computer Engineering
University of Naples
Naples, Italy
paomares@unina.it

Abstract: Application software nowadays tends to be more intelligent to perform actions autonomously, and the development of such software tends to have shorter turn around time. In our approach for designing distributed intelligence systems, each object called a Tele-Action Object (TAO) is enhanced by an index cell (IC). Objects enhanced by index cells can perform actions by themselves. Therefore intelligence is distributed to these tele-action objects. An IC system is an active index consisting of a network of index cells that embodies a lot of distributed knowledge to enhance the overall intelligence of the application software. In the rapid prototyping of time-critical applications, intelligent application software can be generated by combining the technologies of IC cards, IC system (active index), relational mining and time management to transform IC card specifications into executable codes, thus accelerating and simplifying the development of time-critical applications. The Methodology for this transformation approach is described in detail in this paper.

1. Introduction

Recent advances in communications technology, web-based applications and service oriented architecture have stimulated the need for application software development with shorter and shorter turn-around time. Nowadays customers require application software to possess the following characteristics: (a) it can integrate web services, components and legacy software into a functioning system; (b) it must be operational before deadline and remain operational until expiration time; and (c) it must satisfy user-specified timing constraints. These characteristics indicate that application software should be intelligent and capable of performing autonomous actions.

In our approach for the rapid prototyping of distributed intelligence systems, each object is called a Tele-Action Object (TAO), which is a multimedia object with associated hyper-graph structure and knowledge structure [4]. A Tele-Action Object can be as simple as a single piece of information without connection or relation to any other objects. Or we can combine several TAOs in certain connections into a new complex TAO and/or add certain knowledge to a TAO. Basically the TAO is further refined as two parts (G, K): hyper-graph G and knowledge K. For a TAO the hyper-graph G is used to describe the connections and relations between the sub-TAOs within it. The knowledge K is used to describe the actions.
action objects. Objects may also contain multimedia data. An IC system is an active index [2, 3] consisting of a network of index cells that embodies a lot of distributed knowledge to enhance the overall intelligence of the application software.

Based upon index cells and tele-action objects, in the rapid prototyping of time-critical applications we propose an approach by combining the following technologies:

- IC cards,
- IC system (active index),
- Relational Mining for appropriate web services,
- Time Management Techniques.

The IC cards enable the visual specification of an application. It captures the interaction patterns and timing constraints. The interaction patterns depict how the objects interact with each other, and the timing constraints indicate how much time the objects have for action before deadline. The interaction patterns lead to relational graphs specification and protocols, finally the transformation into IC system (active index). The timing constraints lead to time analysis based upon Petri net. The intelligence of ICs in IC system enables the ICs to automatically locate the agents they want to communicate with, and this relational mining mechanism can be used to discover appropriate web services. In component-based software engineering, “gap fulfillment” of the right components uses a similar approach.

The rapid prototyping environment and tools are illustrated by Figure 1.1. The user/developer first creates and edits the IC cards using the IC Card Management System, which generates an XML specification XMLicc. Next the user/developer can design an IC system based upon the user requirements specified by XMLicc using the Multimedia Knowledge Eclipse Environment, which in turn produces another XML specification XMLicx. The IC Software Engineering Environment accepts the specification of the IC system XMLicx, compiles it into executable code, runs the code and produces runtime snapshots for tracing the execution. Since traceability is maintained by the rapid prototyping environment, the user/developer can go back to change the requirements by modifying the IC cards, redesign the IC system, and test it again. Using the above tools the rapid prototyping and development of time-critical applications becomes easier and more effective.

The paper is organized as follows. In Section 2 we present the IC card structure and its XMLicc schema. In Section 3 the visual editor for the IC system, MKEE, is described. In Section 4 we show where the XMLicc schema and XMLicx schema differ and how to transform one schema into the other. The compilation of the XMLicx of an IC system into codes is explained in Section 5. Section 6 describes the IC software engineering environment. In Section 7 we discuss further research topics.

2. IC Card and its XML schema

An IC card is the user’s or developer’s specification of an active object or an agent [5]. An active object usually interacts with other active objects according to certain interaction patterns. There are six basic interaction patterns – quiet (meaning this active object has no tasks and has no interactions with other active objects), by-myself-no-interaction, by-myself-with-interaction, by-others-no-interaction, by-others-with-interaction, and mixed (meaning both active objects have to do tasks and they have interactions). Figure 2.1 shows an example of defining an active object using IC card [5], in which interaction pattern and timing constraint are included.
The IC Card Management System enables efficient editing, organization, and management of IC cards. It maintains a list of \textit{icCardEntry}, which is a collection of \textit{icCard}. Each \textit{icCardEntry} has \textit{icEntryName}, along with EntryId indicating which group an \textit{icCard} belongs to.

For each \textit{icCard}, it includes the following attributes:

- \textit{icName}: name of the IC
- \textit{icId}: id of the IC
- \textit{icDescription}: description for the IC
- \textit{icIntPattern}: how current IC interacts with another IC, can be one of the six patterns
- \textit{icMyTask}: task of the current IC
- \textit{icTimeCriticalCondition}: the timing constraints imposed on the IC
- \textit{icNumberTotal}: N, the total number of IC cards to describe current IC
- \textit{icNumberCurrent}: i, the ith IC card (if N IC cards are used to describe the IC)

\textit{icCard} also has a sub-element \textit{icOther}, which keeps the needed information to interact with another IC. It includes such attributes:

- otherId: the id of the other IC which the current IC will communicate with
- otherName: the name of the other IC
- otherMessage: the message sent to the other IC
- otherTask: the other IC’s task

\section{The Visual Editor for IC System}

The MKEE (Multimedia Knowledge Eclipse Environment), can be found at http://eclipse.dis.unina.it/MkeeSite/ is an Eclipse Development Environment that provides the modeling of multimedia applications and the sharing of knowledge owned by multimedia objects. A multimedia application can be modeled in terms of intelligent objects, i.e., the TAOs [4], which are tele-action objects related by hyper-graph G and enhanced by knowledge K. Every object reacts differently depending on the input that it receives from the outside. The mechanism of answering to the stimuli can be realized by associating a private knowledge to the TAO through the Index Cells [2]. Therefore the application software constructed using this approach supports both the static description of the multimedia application in terms of TAO objects, and the definition of Index Cells net representing the dynamics structure.

A typical IC system is made up of a series of interacting Index Cells, which communicate with each other through message passing. A typical Index Cell has such structure as shown in Figure 3.1:

![Index Cell](image)

\textbf{Figure 3.1. The Index Cell Structure.}

The Index Cell is a particular Finite State Machine which accepts Input messages, executes operations and sends one or more output messages to one or more IC or to external environment. The amount and type (or types) of IC depends on the state and Input Messages. It is a Mealy model machine and, according to the problem domain, could be deterministic or non-deterministic, but as theory states, any ND-FSM could be transformed in a deterministic FSM.

An example of the visual specification of an IC system based on the healthcare application is illustrated in Figure 3.2. In this IC system, Camera captures patient’s images and Sensor captures patient’s health conditions such as blood pressure and temperature. A disabled
patient cannot make alert request by himself/herself, so an alert is initiated by Sensor and/or Camera. Emergency Alert forwards the alert information to Hospital Response when any parameter’s threshold is reached. Then both the Doctor and Nurse will be informed. Nurse will be dispatched to assist the disabled patient if necessary, and Nurse can communicate with Doctor.

![Image](image1.png)

Figure 3.2. A Disabled Patient IC System.

MKEE has the advantage to be a hardware/software platform independent and enable designer to speedily generate application. These advantages are very important in the healthcare environment where a lot of hardware devices and interface to manage there exists. The MKEE tool generates XMLicx specifications of the IC system. Therefore it can be used to serve as the front end for the IC system compiler.

4. Mapping between IC Cards and MKEE IC System

In MKEE the Index Cell is a little different from the IC depicted by IC card. While mapping the same name and id from IC card to Index Cell, Index Cell provides more details such as states and transitions than IC card does. Although the visual representations of IC card Management System and the MKEE IC System are different, they both can be represented by XML, which facilitates the sharing of data across different information systems. Figure 4.1 shows the tree-structured XML schemas used in IC card and IC system.

![Image](image2.png)

Figure 4.1. IC Card XML and IC System XML.

The mapping between IC card in IC Card Management System and Index Cell in MKEE IC system is shown in Table 4.1. The matching shows that the two specifications are indeed compatible.

![Table](table.png)

<table>
<thead>
<tr>
<th>IC Card Management System</th>
<th>MKEE IC System</th>
</tr>
</thead>
<tbody>
<tr>
<td>icCardType.icId</td>
<td>IndexCell.id</td>
</tr>
<tr>
<td>icCardType.icName</td>
<td>IndexCell.name</td>
</tr>
<tr>
<td>icOtherType.icOtherMessage</td>
<td>Message.name</td>
</tr>
<tr>
<td></td>
<td>(The message sent to target IC)</td>
</tr>
<tr>
<td>icOtherType.icOtherName</td>
<td>IndexCell.name of other IC</td>
</tr>
<tr>
<td></td>
<td>(The target IC name of output message)</td>
</tr>
<tr>
<td>icCardType.icMyTask/icOtherType.icOtherTask/icCardType.IntPattern</td>
<td>Relate to the action target in transition (external/source)</td>
</tr>
<tr>
<td>icCardType.icTimeCriticalCondition</td>
<td>Parameter.dataValue</td>
</tr>
<tr>
<td></td>
<td>(Time parameter in messages)</td>
</tr>
</tbody>
</table>

Specifically for the timing constraints, users can indicate the time during which a task should be done in an IC card’s TimeCriticalCondition field. In an IC system, similar timing constraints can be imposed through the parameter of message in transition.

Since both IC Card Management System and MKEE are built on Ecore models, we can use IBM Model Transformation Framework (MTF) to implement partial transformations between the two models. MTF provides an extensible rules language that can be used to define what the
transformation should accomplish, and a transformation engine which can interpret the rules in order to perform the transformation. MTF works on models described by a compatible meta-model in order to express the correspondences in a consistent way. The output of MTF transformation is a set of mappings that relate the objects of two models. The user can specify the mapping as a relation that defines the type of mapping that will apply to model class instances.

5. Compiling IC Specifications

Modern compilers are typically made up of five conceptual phases. The first three phases, lexical analysis, syntax analysis, and semantic analysis, are grouped together to form the compiler’s front end. The fourth conceptual phase is code optimization and is referred to as the middle end. The final phase, dubbed the back end, is the code generator.

The front end is responsible for recognizing validity, or lack thereof, of source language. It also shapes the source language into an intermediate representation (IR), typically an abstract syntax tree (AST) for the middle and back ends. In the case of the IC system, the MKEE visual editor creates the AST. The AST is then analyzed for proper semantics by the compiler. Upon completion of semantic analysis, the task of the front end is complete.

The middle end of the compiler analyzes and transforms the IR through optimizations, so as to improve code quality such as reduced execution time. The IC system compiler does not currently perform any optimizations.

The back end generates the final output in the target language. Contrary to the front end, the back end is source language independent and target language dependent. The IC system compiler generates Java classes, which can be used in cross-platform web applications.

5.1. Input to the IC System Compiler

The MKEE tool is an Eclipse visual editor for specifying IC system. It outputs an XMLicx file describing an IC system. XML documents, due to their inherit tree structure, provide an obvious mapping to an AST in a typical compiler. Once the XML file is created, it can be processed by the IC system compiler. The compiler reads in the XML file, and creates an internal IR. The MKEE tool outputs valid XML files. However, the XML may not be semantically correct for code generation. Semantic analysis is performed on the IR, and, pending valid semantics, output as a set of Java classes. The Java classes can in turn be used in web based Java Server Pages (JSP) applications. When a JSP page is accessed, a web page is dynamically created by compiling the Java classes used by the page. The use of server side computation and the Java programming language allows for platform independence and rapid development of web applications.

5.2. Semantic Analysis

The top level of the IC system is a collection of Index Cells. Index Cells are composed of States and Transitions. For type checking purposes, all immediate children of an Index Cell must be of type State or Transition. States themselves must have a type of internal, entering, or ending. A value of entering indicates that this is a start state. All Index Cells must have exactly one State of type entering. There can be multiple ending and internal states. Each Index Cell must also have a maximum lifetime value. Index Cell’s which do not persist forever will have a numeric value specifying the lifetime in milliseconds. Once the length of the lifetime has expired, the Index Cell will die.

Transitions are hierarchical objects composed of Actions, Input Messages, and Output Messages. Transitions have a source state and a target state. Both the source state and the target state must be resolvable to valid State structures. Transitions also must have a type of boundary or internal. Internal transitions occur between states in the same Index Cell. Boundary transitions occur between states in different Index Cells and are used to pass messages. Internal transitions
are used to drive the progression between States in an Index Cell.

Messages and Actions are composed of Parameters. Parameters must have a data type and data value, as well as a name property. Output Messages have a Target Index Cell that must be resolvable to a valid Index Cell. Actions have simple semantics, requiring only a name.

5.3. Code Generation

The code generator of the IC system compiler creates a set of Java classes that can be used in JSP applications. The code generator is based on a set of transformation rules. The following example illustrates the transformation from MKEE formatted XML to Java code. It is worth mentioning that names in MKEE are entered by the user. The user-provided names are transformed to ensure that they are valid identifiers in the Java language. A simple Index Cell taken from the previously described disabled patient IC system is shown in Figure 5.1. The visualization of the Index Cell will result in the MKEE XML in Listing 5.1. The MKEE XML is passed to the compiler. The compiler will apply a set of transformation rules to the XML resulting in Java source code. Table 5.1 contains a non-exhaustive listing of transformation rules used in the compiler. Transforming most of the XML is a relatively straightforward process. Transitions provide the most challenging transformation process. The Transitions along with the States are transformed into the Index Cell’s transition function. The Java implementation of the transition function for the example Index Cell is shown in Listing 5.2.

<indexCell
currentState="/@icSystem/@indexCell.0/@state.0" id="ic1"
maxLifeTime="infinity" name="Camera">
  <state name="state1-1"/>
  <state name="state1-2"/>
  <transition id="trans1"
source="/@icSystem/@indexCell.0/@state.0"
target="/@icSystem/@indexCell.0/@state.1">
    <message xsi:type="ic:OutputMessage" id="msg1"
name="patient's image">
      <targetIC>ic3</targetIC>
      <parameter dataType="time" dataValue="Tc"
name="Tc"/>
    </message>
    <action body="collect image info periodically" id="act1"
target="source" name="msg1" targetIC="ic3"/>
  </transition>
</indexCell>

Listing 5.1. Specification of an Index Cell.

Table 5.1. Example Transformation Rules.

<table>
<thead>
<tr>
<th>MKEE XML</th>
<th>Java Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;indexCell id=&quot;α&quot;&gt;</td>
<td>public class α</td>
</tr>
<tr>
<td>currentState=&quot;β&quot;&gt;</td>
<td>extends IndexCell</td>
</tr>
<tr>
<td>&lt;parameter dataType=&quot;τ&quot;&gt;</td>
<td>new τ()</td>
</tr>
<tr>
<td>&lt;action id=&quot;ψ&quot;&gt;</td>
<td>private void ψ()</td>
</tr>
<tr>
<td>&lt;targetIC&gt;ψ&lt;/targetIC&gt;</td>
<td>Ψ</td>
</tr>
<tr>
<td>&lt;message id=&quot;ω&quot;&gt;</td>
<td>new ω ( &quot;α&quot;, &quot;ψ&quot;, τ )</td>
</tr>
</tbody>
</table>

public void transition() {
  switch (this.currentState) {
    case icSystemindexCell0state0:  
      changeState ( icSystemindexCell0state1 );  
      break;
    case icSystemindexCell0state1:  
      if (this.previousState ==  
        icSystemindexCell0state0) {
        act1();  
        postMessage( new msg1( "ic1", "ic3",  
          new Object() ) );
      }
      else stateError();  
      break;
    default:  
      stateError();
    }
  }
}

Listing 5.2. Example Transition Function.

In the target code, each Index Cell is implemented as a separate Java class. Each Index Cell class is a thread, in order to allow multiple Index Cells to run concurrently. The States of an Index Cell are implemented as a finite state machine based on the Transitions between them. Messages are implemented as objects which are passed between Index Cells.
The Messages’ Parameters are implemented as the objects’ data fields with corresponding inspector functions. Actions are implemented as function calls. An Action’s Parameters correspond to the arguments to the function. Each Index Cell transitions through its state machine until its lifetime expires. On state transitions, any corresponding output messages and actions are executed. If the next state expects an Input Message, the thread will sleep until it receives any messages it is expecting. Finally, the next transition will be taken.

The code generator can be enhanced by relational mining as follows. If through the relational mining technique an appropriate web service is found to perform certain functions, then the code generator will only produce the Java class to invoke the available web service.

6. The IC Software Engineering Environment

The IC Software Engineering Environment (ICSEE) is an environment for compiling and generating Java codes from MKEE XML output. Figure 6.1 illustrates the interface of ICSEE. The ICSEE main screen shows the various stages including UploadXML, Parsing, Generate, Compile, Load and Instantiate, as well as the options of Save, Delete, Logout, and ReadMe. We will also use the disabled patient example here. Figure 6.1 illustrates the results after UploadXML, showing the various Index Cells.

After uploading the disabled patient XML which is generated by MKEE, click the “Parsing” button. Then all the Index Cells including Camera, Sensor, Sensor Emergency Alert, Hospital Response, Expert, and Nurse will be parsed and displayed in “Index Card Information” section. The Java source code will be generated at the server side when user clicks “Generate”. The server will start compiling the generated Java source code, and the compiled classes will be saved. When clicking “Load”, the generated classes will be loaded into JVM; when clicking “Instantiate”, the created instances will be displayed in the “Index Cell Instances” section.

Using ICSEE the user/developer can easily test and trace the rapid prototyping process. The generated Java code also lays the foundation for further developing application software.

7. Discussion

The transformation approach for rapid prototyping of intelligent software applications is described in this paper. The transformation from IC systems into codes enables the user/developer to explore the design space based upon different IC systems. However the transformation from IC cards to IC systems is still at best a manual process. Our next objective should be to (partially) automate this transformation.

There are different ways to realize the transformation from IC cards to IC system: developing syntactic transformation on the XML documents, or developing graph transformation algorithms to produce a target graph from the initial graph, or using some AI techniques such as rule-based approach. Following the first approach, now we can find the matching between IC cards and IC system and define the mapping rules between the two models. However the IC cards represent an initial specification without details about interaction protocols, timing considerations and protocols. The mapping rules are incompletely defined. Therefore the other approaches are still necessary and require further investigation.

Acknowledgements:

Giuseppe Marco Scarfogliero and Lorenzo Sorrentino implemented the initial version of MKEE, the IC System Visual Editor. Dan Li implemented the software engineering environment (ICSEE) for the IC System Compiler. Robert Zaremba provided the details on the implementation of remote IC subsystems. Their contributions are acknowledged.

References:


Figure 6.1. Results after UploadXML.