Towards Automatic Code Generation for EAI Solutions using DSL Tools *

Hassan A. Sleiman[1], Abdul W. Sultán[2], Rafael Z. Frantz[3], Rafael Corchuelo[1]

[1] Universidad de Sevilla, ETSI Informática
Avda. Reina Mercedes, s/n. Sevilla 41012
hassansleiman@us.es, corchu@us.es
[2] Sytia Informática S.L.
Avda San Sebastián, nº10, Local 1, Entreplanta. Huelva 21004
awsultan@sytiainformatica.es
[3] Universidade Regional do Noroeste do Estado do Rio Grande do Sul (Unijuí)
Departamento de Tecnologia, São Francisco, 501. Ijuí 98700-000 RS Brasil
rzfrantz@unijui.edu.br

Abstract. Current companies count on heterogeneous information technology applications to perform their activities. More often than not, they need to be integrated so that the data they manage is kept in sync or to implement new functionality. According to a recent report by IBM, companies spend from $5 to $20 on integration per dollar spent on developing new functionality. This ratio argues for engineering solutions. The Model-Driven Architecture initiative promotes the development of software systems at different levels of abstraction, and Domain Specific Languages (DSLs) play a prominent role to reduce development costs. By means of an appropriate DSL, software engineers can design a software system that can later be deployed to a variety of specific platforms using automatic transformations. Our proposal to reduce integration costs is a DSL called Guaraná and a software tool to design and automatically deploy integration solutions. Compared to the Enterprise Application Integration UML-profiles, DSLs are more suitable to address problems in a particular domain and are a better approach towards MDA.

Key words: Enterprise Application Integration, Domain Specific Language.

1 Introduction

We define an integration solution as a piece of software that co-ordinates a number of applications exogenously. An integration solution may focus on: Enterprise Application Integration (EAI); Business to Business Integration (B2BI); Enterprise Information Integration (EII); Extract, Transform and Load (ETL) and Mashup. Our research focuses on EAI.

EAI always happens inside the same company, in its own software ecosystem, and the goal is to keep applications synchronized or to create new functionalities on the top

* Partially funded by the Spanish National R&D&I Plan under grant TIN2007-64119, and the Andalusian Local Government under grants P07-TIC-02602, and P08-TIC-4100. The work by R.Z. Frantz was also funded by the Evangelischer Entwicklungsdienst e.V. (EED).

of them. Furthermore, when considering Business to Business Integration, the solution necessarily involves applications that belong to different companies, so different software ecosystems must be considered. This type of solution has the same aim of EAI, but other aspects when building the integration solution must be considered, e.g. authentication, external network communication failures, service availability, confidentiality, non repudiation or accountability must be taken into account. In addition, standards like Open Buying on the Internet (OBI), Electronic Document Interchange (EDI) and Commerce XML (cXML) are usually adopted for the communication amongst companies. Also, it is very important to point here that those pieces of the integration solution deployed inside a software ecosystem (or in case we deploy it as a whole, so the whole solution) will have unlimited access to the application's layers that are part of the same ecosystem; to communicate with external applications that take part of the integration solution, but belong to other companies' software ecosystems, it must be done only by means of the exposed public interfaces of those external applications.

According to a recent report, for each dollar spent on developing an application, companies usually spend from 5 to 20 dollars to integrate it [11]. This claims for solutions that can contribute to reduce this high cost of integration. Guaraná is a domain specific language to design integration solutions that aims to simplify the designing process of solutions and produces platform independent models which can programmatically be transformed into code of a specific technology [6]. In this paper, a realistic integration problem was taken as motivation and validation to generate an application integration solution using Guaraná's language and Windows Workflow Foundation (WF) [4] as target deployment technology. In order to meet this goal we have created a software tool prototype that implements domain specific concepts defined in Guaraná and thus allow the visual design of integration solutions. The platform-independent model designed with this software can be transformed into several integration technologies; however, in this paper, we focus on Windows Workflow Foundation.

Actually, [7] describes patterns for EAI but no integration solution language is defined, while Camel and Biztalk provide languages to describe integration solutions. In the case of Camel, this language is textual whereas Biztalk, although it is a graphical language, its not based on MDA. We understand that a DSL is a well focused language developed to address problems in a particular domain providing a set of dedicated abstractions, elements and notations with formalisation to assist the designer in expressing its solution in the idiom and at the level of abstraction of its DSL. We are aware of the existence of an EAI UML-profile [9] proposed by OMG as an extension for their UML to support some concepts of EAI. While DSL is usually a small and well focused language, UML-profiles intend to wide the scope of UML to cover the modelling of those specific aspects not covered by the native UML elements [1]. This extension has some limitations, such as the impossibility to introduce new modeling concepts that cannot be expressed by extending existing UML elements. These characteristics contributes to make UML-profiles more complex and difficult to use in some domains. This profile also seems to be discontinued.

The transformation of an integration solution designed using Guaraná into Windows Workflow Foundation (WF) is just one among many options, since there are many other possible target integration technologies like those discussed by authors in [5]. These au-

thors also have designed a framework that can be used to compare different integration technologies and they provide a comparison of five technologies in [3]. In [6] authors use 15 properties of this comparison framework to evaluate the Guaraná's language against some most common integration technologies' language.

This paper is organised as follows. Section 2, introduces the domain specific language tools concept and briefly highlights important concepts of Guaraná; Section 3, presents an integration problem from the University of Ijui (UNIJUÍ) used to validate our software tool; Section 4, presents the software tool prototype; Section 5, introduces how the transformation process to Windows Workflow Foundation is carried out; and, finally, in Section 6, we draw our conclusions.

2 Guaraná, a DSL for EAI

Domain Specific Languages (DSLs) are modelling languages designed to be used in special types of problems. They are restricted to a domain and are characterised by a high-level of abstraction that allows to express the concepts in the language of the problem domain. Some examples of traditionally used DSLs include: SQL (for database access), HTML (to describe the structure and content of a document) and BNF (to describe context-free grammars).

The DSLs listed above are textual languages that are specified by describing the language syntax (eg. using BNF), which requires a parser for the language. There are also graphical DSLs based on a set of graphic symbols. Such DSLs have a direct correspondence with the conceptual model. Besides, the description of these languages can be done graphically. Considering the benefits of graphical DSLs, we have relied on this kind of DSL to perform this work.

The most important aspects of a graphical DSL are: domain model, notation and code generation. The domain model is a model of concepts described by the language. The basic constructors in graphical DSLs are domain classes and domain relationships. Domain classes represent the concepts of the problem domain whereas domain relationships represent relationships between these concepts. In addition, each domain class and domain relationship contains a set of properties. An important aspect of the domain model is the definition of constraints that the model must satisfy. These constraints are used to verify that the created diagrams are valid. The notation is a set of graphical symbols used to represent the domain model. The basic elements are shapes and connectors that are the graphical representation of the domain classes and the domain relationships. There is a direct correspondence between domain classes and domain relations and its graphical representation. This graphical notation is used to edit the model.

2.1 Guaraná

Guaraná provides a set of domain specific constructors to design integration solutions, which are described in the language's metamodel [6], where a part of them are inspired from the patterns at [7]. This language provides a very expressive and needful graphical notation for these constructors, which allow us to visually design an integration solution. Below we introduce the main constructors that can be seen in Fig. 1.

Building Block: This is one of the most important concepts in our language, since it represents a general constructor block where most of an integration solution processing takes place. Although some building blocks receive no entry messages, a typical building block receives an inbound message, executes one or more atomic tasks, and then makes the resulting message available for the next element(s)in the flow. Apart from being composed of tasks, building blocks have ports through which they receive and send messages. Basically, there are two kinds of building blocks: Wrappers and Processes. Wrappers are used to connect an application to an integration solution. Therefore, necessarily, one of its ports are connected with a specific application. Its internal tasks prepares messages to be sent to the application and/or to other processes of the integration solution. On the other hand, processes represent internal blocks where a well-defined set of tasks perform a clear integration service of the solution. A process is connected to other processes or wrappers by means of ports and integration links.

Task: Is the element responsible for a building block's internal processing and allows to turn a process or wrapper into a more complex processing unit. A task reads a message from a slot, processes it and writes the result to the next slot, making it available for the next element inside the block. This message processing usually consists of executing an integration pattern, e.g. enriching, translating, filtering or routing.

Slot: They are used inside building blocks to allow exchanging messages between ports and tasks, and also between tasks,ie., they are essentially buffers.

Port: These elements abstract away from the method used to communicate building blocks and/or applications. Guaraná provides four types of ports, divided into two categories: one-way ports and two-way ports. The former are used for internal or external communication whereas the latter are used for external communication with an application. One-way ports are divided into entry port and exit port; two-way ports are divided into solicit-responde port (solicitor port) and request-response port (responder port). An entry port reads information from an application by accessing one or more of its layers. The possible layers we consider here are: data layer, business layer, controller layer and graphical user interface layer. An exit port does the opposite of an entry port, that is, it writes information to an application's layer. A solicitor port enables the integration solution to solicit information from an application. A responder port provides a request-response interface that an application can use to send requesting messages and receiving responses from the integration solution.

Integration Link: Used internally in an integration solution as a mean to transport messages from one building block to another. Because ports represent entry/exit points at a building block, integration links are those elements that actually connect them.

3 Validation example

We have validated our proposal by designing and generating code for an integration problem found at UNIJUÍ. The goal was to automate the invoicing of the calls not related to te employee's work activities. At UNIJUÍ, five applications involved in a handcrafted process to invoice their employees of the private phone calls they make using the University's phones. Applications run on a different platform and were designed without integration concerns in mind. There is a Call Center System (CCS) that records

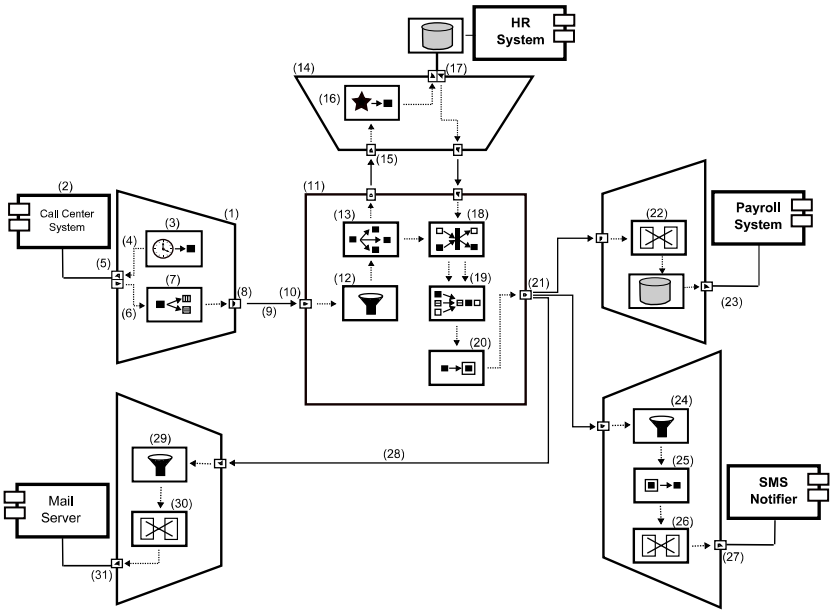

Fig. 1. UNIJUÍ's integration solution.

every call every employee makes from one of the telephones this university provides to them. Every month, an analysis is performed to find out the price of the private phone calls; such calls are debited to the employees by using a Payroll System (PS). There is also a Human Resources System (HRS) that provides personal information about the employees, including their names, phone numbers, social security numbers, and so forth. There are two additional systems, a Mail Server (MS) and an SMS Notifier. Figure 1 depicts these applications and our integration solution.

The integration flow for the solution presented in this example begins with a timer task (3) inside the wrapper (1) of application CCS (2). This task creates an activation message every five minutes and writes it to the slot (4) that connects it to the solicitor port (5), which then is accessing. This message activates the solicitor port that extracts all those calls which were made in the last five minutes. The only way to communicate with this application is by means of its user interface since no clean interface is provided, so the solicitor port will have to use a scrapper [2] to perform its work. The port will return a big message that contains, probably, many phone calls and then writes it to the next slot (6) in the flow. At this point a splitter task (7) is used to break the incoming message into new messages that contains just one phone call each. The exit port (8) of this wrapper reads each message from its previous slot and then sends it to the integration link (9), making it available for the entry port (10) of the central process block (11). This process starts with a filtering task (12)which filters out messages that do not have a cost for the university, and allow just toll calls to remain in the flow. Those messages are written to the next slot, and will be read by the next task, a replicator (13). The replicator makes copies of the original message. In this case one copy is sent to the wrapper of the application HRS and the other to the next element in the current process. In this integration solution we need to append missing information about the employee to the message, like: name, department, e-mail and mobile phone, and for this purpose the HRS, that contain this information, is also integrated into our solution. The message

copy received by the wrapper of HRS, through the entry port (15), will be processed by a custom task (16). This task produces an outbound message that represents a database query to be executed against the database using the solicitor port (17) of this wrapper. After that when the correlation set task (18) receives the result from the HRS's wrapper, it gives pass to the result and the original message that was waiting inside one of its entry slots. The next task, a merger (19) reads the two correlated messages and writes them to the single entry slot of the enricher task (20), so it enriches the original message with the result from the HRS. Now the enriched message is sent to the next slot, the one that connects with the exit port (21). This port is also connected to three integration links that allow sending a message copy to PS, SMS and MS.

The first integration flow after the exit port (21) connects the process (11) to the wrapper of the PS application. This wrapper receives the message through its entry port and makes it available for the translator task (22). The translator is responsible for translating the current message format into a new format that the PS can understand, and so the exit port (23) of the wrapper writes the message into the application's database. The second flow connects the same exit port (21) to the SMS' wrapper. This wrapper has a filter task (24) to filter out those messages that, for some reason, does not have the employee's phone number and then those messages that could pass are sent to a slimmer task (25). A slimmer is responsible for removing some information from the message in order to make it smaller before sending it to the SMS. The SMS is an external application that allows sending messages to mobile phones. The last task in this wrapper is a translator (26) that receives the inbound message and translates it into a special format that the SMS can understand. Once the SMS offers a public gateway the exit port (27) will interact with it through remote procedure calls to forward the message.

The last copy of the message goes to the flow (28) that now connects the process (11) with the wrapper of the MS. This wrapper integrates the application allowing the solution to send e-mails with all the details about the employee's call. The first task in this wrapper is a filter (29), here again to filter out messages that for some reason does not have the employee's email address. As in the other wrappers it is important to translate the inbound message into a message format that the MS can understand. This is done using a translator (30) inside the wrapper, just before its exit port (31). The translated message now goes, through a port that uses remote procedure call to communicate with the application's gateway.

4 A DSL tool for EAI

Microsoft DSL Tools (MS/DSL Tools) is a framework that eases the development of DSLs and graphical editors for them. The framework consists of a project wizard to create configured solutions, a graphical designer for defining and editing domain models, designer definitions in XML, a set of code generators that produce code that implements domain model definition, a designer definition and a framework for defining text output generators [8].

Domain models in MS/DSL Tools are defined using class hierarchies and identifying relationships between theses classes. Relationships can be either reference or com-

position. First, we define Guaraná's metamodel and once defined, we proceed to create this metamodel in MS/DSL Tools. IntegrationSolution is the root class of integration metamodel. This class represents the integration solution. An integration solution designed with MS/DSL Tools is an instance of this class.

An Integration Solution is composed of IntegrationSolutionBlocks and Applications. A composition can be represented in MS/DSL Tools by an embedding relationship. There are more composition relationships in the integration metamodel. A block in an integration solution is composed of EntryPort and ExitPort that allow the connection of IntegrationSolutionBlocks which are composed of a set of tasks whose functionalities were described before.

Another kind of relationships is the inheritance. In our integration metamodel, BuildingBlock and Hub are classes that derive from IntegrationSolutionBlock, while Wrapper and Process derive from BuildingBlock. Further, all the kinds of tasks inherit from Task. Unlike composition, reference relationships do not have any limitation on the multiplicities of each of their roles. This allows us to connect any type of class to create graphs with really complex models.

Entry ports and exit ports are always bound with one another through integration link relationship. Applications are connected with their wrappers through application link relationship. Other reference relationships in integration metamodel are slots that interconnect tasks to each other. After creating the above metamodel, shortly explained here, we obtain Guaraná's Designer, cf. Figure 2.

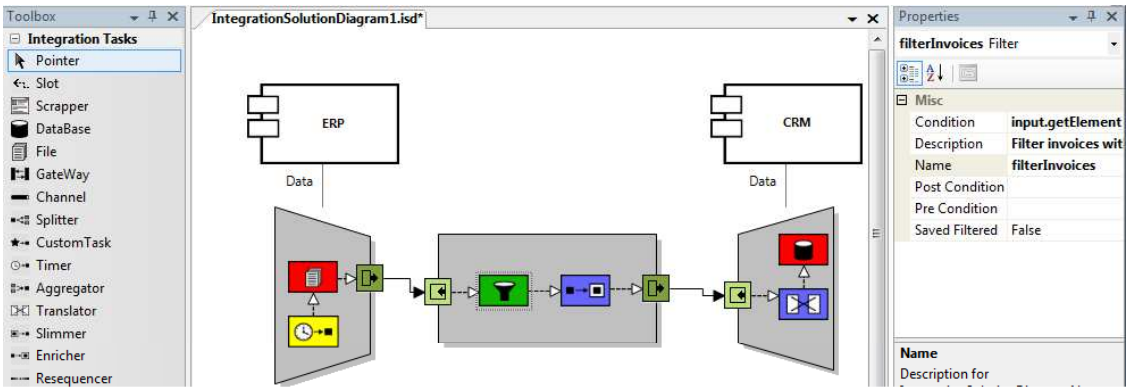

Fig. 2. An integration solution created in Guarana's DSL tools.

Guaraná's Designer consists of a drawing area where integrations solutions can be modelled and a toolbox that allows us to drag and drop elements of Guaraná's DSL in an integration solution. Guaraná's DSL tool has also a properties window where we can edit the properties of ports, tasks and building blocks.

In the toolbox, we can find Process, Application and Wrapper which are elements that can be added directly to the model. Tasks are divided into four groups (Router, Constructor, Interfacing and Transformer) represented by different colors (green, yellow, red and blue respectively). These tasks can only be created within a process or a wrapper. Entry Ports and Exit Ports which also apply to the Processes and Wrappers. We can find three types of connectors: Integration Links to connect with Entry Ports and

Exit Ports, Slots for connecting Tasks and Ports within the Processes and Wrappers, and Application Links to connect applications and wrappers.

5 Automatic code generation

Our proposal is to transform our designed integration solutions into workflows using Microsoft Windows Workflow Foundation (WF), and then make use of the WF's runtime to run the generated workflow. From our point of view, an integration process can be seen as a business process where information flows between participants.

There is no doubt that the appearance of workflows has changed the way processes are done. The partial or complete automating of a business process is called workflow, where documents, information or tasks flow between the participants under some constraints and rules, and where the main goal is to achieve a business goal, such as client satisfaction. We can define workflows as the movement of information through a business process between resources [10]. First, we transform our integration solution into a workflow, then we use WF as a workflow management system in which the generated workflow is executed and monitored.

5.1 Transformating our DSL into WF

Transformation is achieved by the creation of three files: a designing file where the workflow is described (Xoml in this case), a code file where tasks' functionalities are described (C#), and a third file that will host and launch the created workflow (C#). The basic unit in an integration solution is the task, while the basic unit in a workflow is called activity. In this transformation, every task of the integration solution is transformed into one or more workflow activities. The resulting activities represent the functionality of the original task in our integrations solution, but some limitations might not allow us to implement the complete functionality. We start explaining the implemented transformations and then we describe the limitations we found.

System messages: To model system messages, we used XML. A message in our system is an XML file, which we chose for the easiness of its management and the large number of APIs that can be used to manipulate this type of files. It is an instance of the class XmlDocument of the .Net API from Microsoft. Messages inside our system do not have a predefined scheme since they are produced by the applications we are integrating. It is user's responsibility to transform the input messages of an application to another format understandable by another application using a task from the transformers group.

Wrappers: Inside a wrapper, we can find tasks from the interfacing group that are used to communicate with the applications we are integrating, but it can be seen as a process too, where the first activity reads information from an application and then it is processed and written to an exit port.

Processes: An integration process can be seen as a business process. Since WF is oriented to business processes, we were able, although with some limitations, to transform integration processes into workflows. To achieve this, entry and exit ports, tasks and slots that interconnect these tasks were transformed into workflow elements.

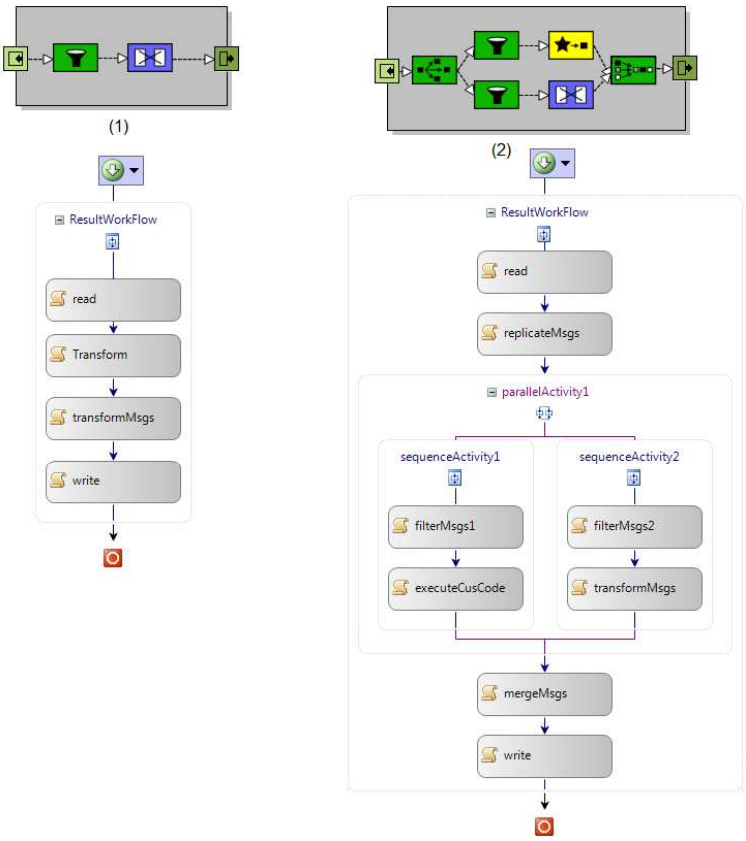

Fig. 3. Integration processes seen as workflows in WF.

Ports: Ports are simulated by activities that read from input directories and write to output directories. An entry port is an activity that reads XML files from a directory to inject them into the process so that they flow between processes's tasks. Exit ports are activities that write messages to an output directory. We used .Net features to monitor directories so that a process can be activated only when one or more messages are detected in the entry port (input directory). When a port reads a message, it is consumed and disappears from the port.

Slots: In an integration solution, a slot is totally different from arrows and links connecting workflow activities. In workflows, an arrow indicates that the origin task is a predecessor and the target task is a successor. For this reason, the responsibility of writing resulting messages in the target tasks were assigned to the task that produces the messages due to this difference in concepts. A predecessor task has a reference to the successor's input list where it writes its output messages.

Tasks: Every task is mapped onto one ore more activities to ensure the compliment of the maximum part of the task's functionality. As we mentioned before, tasks are classified into 4 groups:

Constructors: Custom Tasks are converted into custom activities where user can insert custom code. For this purpose, the user can use some variables we predefined such as input messages to read the incoming messages and the output messages where it can write the resulting messages for the successors tasks. The splitter and its opposite task, the aggregator, were transformed using XPath to implement these tasks' functionalities.

An aggregator or splitter in our integration solution needs Xpath expressions, provided by user, to fragment an input message creating various output messages, or to aggregate the input messages creating a new one. When it is transformed into WF, a WF activity executes this XPath expression over the input messages and generates a unique output message.

Transformers: This type of tasks was implemented using XSL files that transform XML messages from one scheme to another. In the case of the translator, the user should indicate the XSL transformation that must be applied to get the new type of messages. In WF, this is an activity that reads the input message, uses the .Net API to apply XSL transformation over it, and writes the resulting message in the input buffer of the following activities. XSL transformations were also used for the Enricher and the Slimmer, whose functionalities are opposite to each other. In the case of the enricher, the XSL should query a database or some files, creating a new message with the old one and the new information, whereas the slimmer's XSL file creates a new message eliminating unnecessary data from the original message. Transformation of this group of tasks was simple using of the .Net API for XML transformation.

Routers: Filters seem like WF's If-Else activity, but implementing the conditions for this type of activity in WF is not so simple since they should be expressed in a complex XML structure. Another way to see this activity is as any other WF activity that before writing any message to the successor's input buffer, the activity checks if it satisfies the condition entered by user in the Filter task. Some variables are provided to the user so that conditions may be defined over input messages giving conditions more expressiveness. Another task in this group is the replicator, which was simple to implement using the XML API in the .Net to clone messages and place a copy for each successor. The Resequencer was not implemented because the message's model in our implementation doesn't have any identifier that can be used to order the incoming messages. The other two tasks of this group are the Distributor and the Merger. Due to some limitations that we discuss in the next section, the user must write his or her custom code in the case of the distributor, but he can make use of lists of messages where successor's messages are saved. This way, a user can introduce and check the conditions that should be satisfied before passing the messages to a successor task. In the case of the merger, the WF activity reads the XML messages from its input and creates an output message by gathering the input messages under a unique root. A replicator may have more than one output whereas a merger has more than one input. We simulated these tasks making use of the parallel activities of WF.

Interfacing: Neither does WF provide predefined activities, nor facilitates implementing the tasks in this group. These tasks were transformed into activities that call C# code where data bases, files and web services are accessed and information is read. Some scrappers were also implemented to be used inside our designed integration solutions.

5.2 Limitations

Similarities exist between workflows and integration solutions, but depending on the implementation, some features may be used, while other features are not available. In

our case, some of the characteristics of WF helped us, such as the XML language to define workflows, the workflows designer and the ability to execute our C# code inside our activities. Other functionalities were not implemented or were partially implemented.

In the case of the constructors, although WF provides an activity similar to the Timer called Delay Activity, this activity has a different purpose, since a Timer is a task that reads messages at regular time intervals, whereas the delay activity in WF causes a delay in execution of the workflow. Timers were not implemented due to the lack of a similar activity in WF and other limitations we mention in the next section.

In the routers group, a Distributor has a condition for every successor slot, and only writes the message in this slot if the condition is satisfied. Here slots can have conditions, buffer, and other properties. In a workflow, there exists a slight similar concept which is the arrow that interconnects activities. It is used to indicate that the activity connected to the origin is the predecessor of the task connected to the end of this arrow (target activity). This limitation made us implement slots in a different manner, so now it is the predecessor's responsibility to pass messages and to write them in the input buffer of the successors.

More serious limitations are the slot cardinalities. In our integration solution, a slot may have more than one input and more than one output, but for a workflow in WF, an activity has only one input, and a unique output. Tasks with more than one output (Replicator, Distributor, etc.) and tasks with more than one input like Merger, are implemented using parallel activities. In the first case, tasks with multiple outputs are always followed by a parallel activity where each successor activity occupies a branch of this parallel activity, while in the second case, this type of tasks always comes after a parallel activity where each branch of this parallel activity contains a preceding activity of the merger in our integration solution. An example can be seen in the second process of Figure 3, where a replicator is followed by a parallel activity while a merger is put just after a parallel activity.

The limitation that most affected this implementation, was the number of threads. In a workflow, it is supposed that a unique thread moves information and accomplishes tasks, while in our integration solution each task could be a thread. For example, while a task is reading from a database, another can be transforming the information read so far. The simulation of this feature, was done by creating an input directory, while a file system watcher monitors changes in this directory. When messages are created in this directory, an instance of the workflow, result of the transformation, is created and messages are injected to this workflow. The injected messages then flow inside the generated workflow where activities perform tasks' functionalities over these messages.

Using the WF implementation, some characteristics were lost, such as simultaneous tasks execution, multiple inputs, multiple outputs and the interfacing group which was not implemented at all, but other features and advantages were gained like WF services and the simplicity of the definition of workflows in WF. A workflow in WF is thought to implement the business logic inside a user application, helping the creation of Process-driven applications. WF doesn't offer activities to communicate with other applications (only with web services).

6 Conclusions and future work

During the last decades many monolithic, single-purpose applications were created for supporting companies' businesses. Part of these applications, nowadays called legacy systems, are still running in a distributed environment and so with other software packages purchased from third parties and new specially tailored applications, represent the software eco-system found in a large number of companies. The process of integrating applications inside or amongst different companies, known, respectively, as Enterprise Application Integration and Business to Business Integration, is still a very costly task. We believe a domain specific language and appropriate software tools to design platform independent models for integration solutions which can be programmatically transformed into a specific deployment technology, could help to reduce this cost.

In this paper, we introduced Guaraná, its editor and a transformer. We used DSL Tools technology provided by Microsoft to develop a software tool which implements the domain specific language Guaraná, allowing to visually design integration solutions using this language and also to deploy them to WF technology. In this first approximation to a programmatically transformation of platform independent models into executable code, we have also used WF's runtime environment to host and run the deployed integration solution. This experience has provided us sufficiently information to motivate us to improve our software tool by using the Model Driven Architecture approach to perform the transformation process from the platform independent model of an integration solution into executable code of a target technology.

References

1. A. Abouzahra, J. Bézivin, M. D. Del Fabro, and F. Jouault. A practical approach to bridging domain specific languages with UML profiles. In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, volume 5, 2005.
2. C. Chang, M. Kayed, M.R. Girgis, and K.F. Shaalan. Survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006.
3. R. Corchuelo, R.Z. Frantz, and J. González. Una Comparación de ESBs desde la Perspectiva de la Integración de Aplicaciones. In *Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, 2008.
4. Microsoft Corporation. Windows Workflow Foundation Home, 2009.
5. R.Z. Frantz and R. Corchuelo. Integración de aplicaciones: Un lenguaje específico de dominio para el diseño de soluciones de integración. Technical report, Universidad de Sevilla, 2008.
6. R.Z. Frantz, R. Corchuelo, and J. González. Advances in a DSL for Application Integration. In *Proceedings of the Zoco08 Workshop*, pages 54–66, Gijón (España), 2008.
7. Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.
8. Microsoft. Overview of domain-specific language tools, 2009.
9. Object Management Group (OMG). OMG EAI Profile Home, 2004.
10. Charles Plesums. Introduction to workflow. Technical report, Computer Sciences Corporation, Financial Services Group, 2002.
11. J. Weiss. Aligning relationships: Optimizing the value of strategic outsourcing. Technical report, IBM, 2005.