FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

BSopt Choreographies - Transforming Global Choreographies into Workflow Deployment Artifacts

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Mario Topf
Matrikelnummer 0025177

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Mag. Dr. Christian Huemer
Mitwirkung: Univ.Ass. Dipl.-Ing. Mag. Dr. Marco Zapletal

Wien, 27.09.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Mario Topf, Wiedner Hauptstrasse 117, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. September 2010

Mario Topf

Acknowledgments

First and foremost I want to thank my parents, who always supported me over the course of my studies.

I also want to express my thankfulness towards my advisors Christian Huemer and Marco Zapletal, who promptly provided me with guidance when needed during the course of the BSopt project and during the writing of my thesis.

Last but not least, I want to thank my colleagues, who also worked in the context of the BSopt project and consequently strived to make BSopt Designer a reality: Philip Liegl, Dieter Mayrhofer, Thomas Motal, Rainer Schuster and Hannes Werthner.

Figure 0.1: Finished.

Abstract

This thesis covers the transformation of interorganizational business processes in the area of business-to-business (B2B) electronic commerce. It builds upon the approach devised as part of the *Business Semantics on top of Process Technology (BSOPT)* project [2]. BSopt aims to integrate management-, administration- and IT perspectives in B2B collaborations as inspired by the Open-edi reference model [20] into one consistent tool environment. The resulting BSopt Designer [24] tool was extended by the use of visual *domain specific languages (DSLs)* encompassing the concepts given in Open-edi. Additionally, C# developed transformation wizards were created to support a semi-automatic mapping from business models to business process models and, finally, to technologically dependent executable deployment artifacts. The thesis describes the contributions to the BSopt approach starting with the transformation of business process descriptions as seen by a neutral observer into a participant-view dependent design. It continues to show, how these so called "local choreographies" are transformed further into concrete *Microsoft Windows Workflow Foundation (WF) 4.0* based workflow artifacts ready for integration into the *service oriented architecture (SOA)* of IT environments. In doing so we also accommodate adapted business document specifications into the transformation process. These specifications are used in business processes to define which messages are sent or received at any point in time during a B2B collaboration. The last contribution to the BSopt approach comes in the form of an example business application. It supports the one-click hosting of generated workflow artifacts and acts as a tool to test the validity and correct functionality of the generated workflows. An accompanying example scenario involving both buyer and seller in a typical ordering process is introduced early in the thesis. It acts as an ongoing demonstration of the different views on business processes and finally illustrates, how the decisions of the involved parties influence the outcome of the business process in the example business application.

Kurzfassung

Diese Diplomarbeit behandelt die Transformation von Geschäftsprozessen zwischen Unternehmen im Business-to-Business (B2B) Bereich des elektronischen Handels. Sie baut hierzu auf dem im Rahmen des *Business Semantics on top of Process Technology (BSOPT)* [2] Projekts erdachten Ansatzes auf. BSopt zielt darauf ab, die Management-, Verwaltungs- und IT Perspektiven innerhalb von B2B Kollaborationen, wie im Open-edi Referenz Modell [20] vorgestellt, in eine konsistente Toolumgebung zu integrieren. Das hieraus entstandene BSopt Designer [24] Tool wurde um visuelle *domänenspezifische Sprachen (DSLs)* gemäss der Konzepte aus Open-edi erweitert. Zusätzlich wurden in C# entwickelte Transformations-Wizards entwickelt, um halbautomatische Abbildungen von Geschäftsmodellen nach Geschäftsprozessmodellen und schliesslich technologieabhängiger ausführbarer Deployment-Artefakte zu unterstützen. Die Diplomarbeit beschreibt die Beiträge zum BSopt-Ansatz beginnend mit der Transformation von Geschäftsprozessmodellen, beschrieben aus der Perspektive eines externen neutralen Beobachters, in eine Darstellung, die sich an der Perspektive eines bestimmten Teilnehmers des Prozesses orientiert. Sie stellt weiters dar, wie diese sogenannten "lokalen Choreographien" weiter in *Microsoft Windows Workflow Foundation (WF) 4.0* basierte Artefakte transformiert werden, welche zudem direkt in die *serviceorientierte Architektur (SOA)* einer IT Umgebung integriert werden können. Innerhalb dieses Schrittes nehmen wir zusätzlich angepasste Spezifikationen von Geschäftsdokumenten im Transformationsprozess auf. Diese Spezifikationen werden im Geschäftsprozess verwendet, um zu definieren, welche Nachrichten zu jedwedem Zeitpunkt während einer B2B Kollaboration gesendet oder empfangen werden. Der letzte Beitrag für den BSopt-Ansatz besteht aus einer Beispielsgeschäftsapplikation. Diese unterstützt das Ein-Klick Hosten der erzeugten Workflow Artefakte und fungiert als Werkzeug um die Validität und korrekte Funktionalität der erzeugten Workflows sicherzustellen. Ein begleitendes Beispielszenario, welches Käufer und Verkäufer in einem typischen Bestellsprozess darstellt, wird zusätzlich in der Arbeit vorgestellt. Dieses agiert als laufende Demonstration der unterschiedlichen Ansichten auf Geschäftsprozesse und illustriert schlussendlich, wie die Entscheidungen der eingebundenen Parteien den Ausgang des Geschäftsprozesses in der Beipielsgeschäftsapplikation beeinflussen.

Inhaltsverzeichnis

1 Introduction

The area of e-commerce has been in a steady development since the late 1970ies. Doing business by exchanging electronic documents in contrast to information printed and delivered on paper promises great advantages in speed and manageability. On the other hand information processed by computer systems had to be defined unambiguously which required the introduction of structured data and mutually agreed upon data exchange standards. E-commerce and more specifically the area of business to business (B2B) were long inhibited by the resulting document standards which suffered from companies exposing their data in ambivalent ways and the usage of different technologies for delivery. This lead to a situation where only few enterprises willing and able to invest heavily into the area then called *Electronic Data Interchange (EDI)* could cope with the ensuing management requirements of those times. After the commercial usage of the Internet as inexpensive means for a worldwide communication network was allowed in 1991, companies slowly started to see its potential. This facilitated a new boom in the B2B area as smaller enterprises saw their chance to profit from the new technological possibilities. While a new communication platform and the advent of related technologies such as XML could not solve the problem of distinct data interpretation, progress went on and brought with it new ways to automate enterprises. EDI first had a document centric view with employees knowing how to handle incoming data and their linked semantic meaning in an overall business process. Those processes were typically based on a set of rules which were implicitly applied by the officials assigned to deal with the incoming information. Enterprises started to uncover those structures and use the ideas of business process modeling to explicitly define the logic needed to reach their business goals in a B2B context and to be able to assess its efficiency.

As the human factor was gradually replaced by automated processes, companies no longer depend on interpersonal agreements to reach their business goals. In return the awareness of collaborating parties about a common way of predictable electronic correspondence is required. Businesses need to publish which business processes they support in order to be able to reach other players on the market with their service offerings. The service as centerpiece in the execution of a business collaboration has also established itself in the paradigm of service-oriented architectures (SOA) which evolved as a widely adopted methodology for service-based communication. This paradigm allows the realization of business processes by aligning services with the desired procedure. More generally SOA according to [49] is about *örganizing and utilizing distributed capabilities that may be under control of different ownership domains. In general, entities (people and organizations) create capabilities to solve or support a solution for the problems they face in the course of their business"*. Furthermore, it is concretized: *"The main drivers for SOA-based architectures are to facilitate the manageable growth of large-scale enterprise systems, to facilitate Internet-scale provisioning and use of services and to reduce costs in organization to organization cooperation"*. It follows that SOA not only encompasses IT realizations such as the common adoption of web services but is deeply linked with the requirements of the business it aims to support.

The Open-edi reference model coincides with SOA on this matter. The ISO standard 14662 [20] on the Open-edi reference model partitions B2B related concerns into two

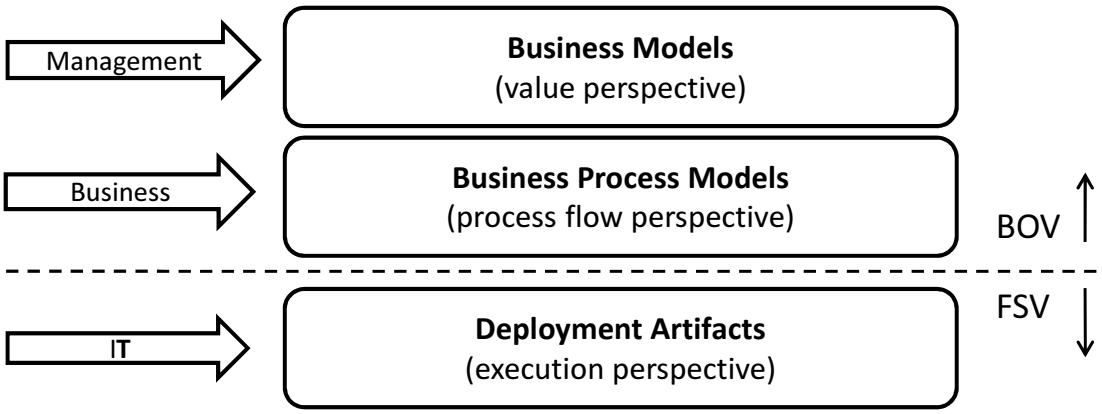

Abbildung 1.1: The three distinct perspectives of B2B collaborations

main views as illustrated in figure 1.1. The business operational view (BOV) encompasses the business side which itself consists of a value- and a process flow perspective. The value perspective in the BOV is concerned with a business model which defines the basis for actions in e-commerce. After all only business models which provide possibilities for seizing profit are viable options. The second perspective included in the BOV is based on business process models which define the sequence of interactions between business partners for a given business model, the structure of interchanged information and all possible outcomes of such processes. All information defined in the BOV is technology independent. This is in contrast to the functional service view (FSV), the second big aspect in Open-edi which is concerned with concrete technology suited to implement the information coming from the BOV. The deployment artifacts resulting from the execution perspective in the FSV are derivations from the information given in the BOV and thus different technological solutions for the same business definitions are possible.

This thesis is based on an application implementation carried out for the *Business Semantics on Top of Process Technology (BSopt)* project [2]. It follows the semantics of Open-edi and encompasses the formulation of business models, business process models and deployment artifacts in a top-down wizard driven construction process. In doing so users of BSopt technology are enabled to integrate their business service offerings into their IT environment based on the concepts of SOA.

1.1 Motivation

This section describes motivators which lead the BSopt project to create its main software delivery called *BSopt Designer* [24]. This tool is an integrated environment for the formulation of data in the BOV of Open-edi. Importantly it also supports the semi automatic mapping between those data artifacts.Furthermore it allows the creation of technology dependent deployment artifacts in the FSV by supporting the transformation of business process models. These motivators are important for this thesis as it covers the software modules developed for the BSopt project.

The creation of *BSopt Designer* is tailored to the needs of three different groups of users. Those consist of individuals with either a management-, business modeling-, or IT-background. Traditionally each group comes with different tools of trade to formulate their output. People from management are used to consider the implications of business models by filling spreadsheets to anticipate economic opportunities and threats. Busi-

ness analysts work with specialized tools including those for business process modeling by, e.g., arranging and connecting graphical elements on a canvas. At last developers and system administrators on the IT layer of Open-edi use integrated development environments, administration panels and command line shells to create and handle deployment artifacts representing a business process. Information coming from management must be reinterpreted by business analysts to be formulated as business process. Correspondingly, data formulated in the process flow perspective can not be directly adopted by IT in order to create the desired deployment artifacts. One motivational aspect coming from this background is the heterogeneity of information on each of those layers. As data from the value perspective has to be reinterpreted to be useful for business process modeling (as is the same with process flow perspective data used in the execution perspective) resources are spent on restructuring of data. The *BSopt Designer* tool is made with a design in mind to alleviate this process and allow semi automatic mappings from value perspective data to process flow information and finally readily executable workflow artifacts which can be integrated into an IT environment. This way businesses are able to react in a more efficient way to ever changing business realities by minimizing turnaround times.

Another motivational factor for BSopt comes from the observation that current tools designed to define business processes and deployment artifacts tend to set their sole focus on the technologies necessary to finally support the process. As business processes themselves derive their legitimacy from their ability to reach their company's business goals, the BSopt project team considered the inclusion of facilities to define business models focusing on the value perspective an essential and novel feature.

The derivation of new representations by transforming and enriching data from a given perspective is also in line with the strategy of a "top-down approach". Businesses following the contrary bottom-up approach start with the formulation of their desired business process by defining internal execution semantics and continue with the specification of external message exchange details. This strategy leads to a business process definition with little chance for incidental support by other players on the market. Only big enterprises able to dictate the rules may be successful this way as they may force potential business partners to adopt the given process as centrally defined. In contrast the BSopt project aims to support business collaborations by first defining processes from a neutral global perspective. This view also acts as a service contract for all involved parties and serves as a common source for the derivation of fitting complementary business process descriptions to be adopted by market participants. While all parties still need to agree on a common global version for a business process, each company may implement the internals as of their choosing to satisfy their needs. This top-down approach enables businesses to search for potential partners also supporting the same globally defined business process. By basing the adoption of business processes on a global perspective the fragmentation of the market can be avoided as processes serving a specific business goal come only in few specific formulations. This is in contrast to a scenario where each company would publish their business process descriptions based on their very own bottom-up design.

1.2 Contribution

The contributions for this thesis are all derived from the prospect of transforming a business process model into executable workflow artifacts. The resulting implementation parts were all integrated into the *BSopt Designer* tool environment and consist of transformation dialogs leading the user through corresponding wizard user interfa-

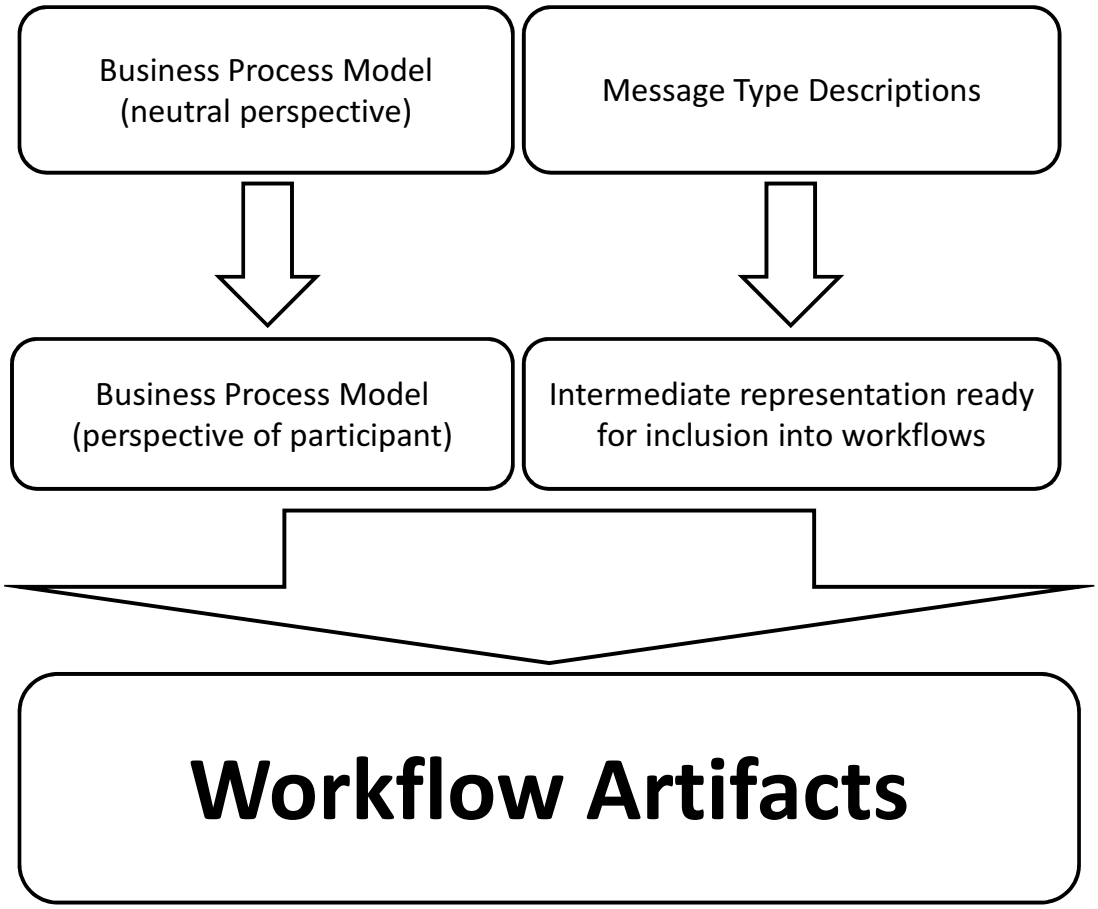

Abbildung 1.2: Steps in the creation process of workflow artifacts

ces and the actual transformation logic. Figure 1.2 shows a generic overview of the accomplished tasks necessary to create workflow artifacts. It illustrates that business process models first describe their specific logic in terms of a neutral perspective. This process definition has to be transformed into definitions describing the process from the perspective of each participating business partner. Additionally the messaging part has to be handled which is accomplished by transforming given message type descriptions outlining which information is to be exchanged in a given business process into a fitting intermediate representation. The final step is to take both a business process model from the perspective of one participant and the message type intermediate representation and transform those into workflow artifacts ready to be hosted by an appropriate business application. In order to test the validity of the workflow artifacts generated an example business application also had to be developed.

Factored into distinct tasks the contributions for this thesis are fourfold:

1. Transform a business process model defined from a neutral perspective into business process models defined from the perspective of each participating business partner.

2. Transform message type descriptions into a fitting intermediate representation suitable for integration into the final workflow artifacts.

3. Transform a business process model from the perspective of a specific participant into workflow artifacts by integrating the intermediate messaging representations

and specific user input into the process.

4. Provide an example business application which is able to host workflow artifacts generated with *BSopt Designer* in order to execute the embedded business logic.

1.3 Structure of the thesis

The remaining parts of this theses are structured as follows: Section 2 covers the two main technologies dealt with in the actual implementation work for *BSopt Designer*: domain specific languages and workflows. It outlines the current state of the art and also familiarizes the reader with the specific technologies used or tested to accomplish the project goals. Section 3 covers the contributions for this thesis in detail. It describes the transformation of business process models and message type descriptions. Additionally it describes the workings of the proof of concept business application, how specific problems with the provided APIs have been solved and accompanies each specific subsection with an ongoing example showcasing how the result can be treated by a user to automatically generate workflow artifacts. Finally, section 4 briefly summarizes the thesis, ending with a conclusion and offers an outlook into future work related to the approaches in the BSopt project.

2 State of The Art and Selected Technologies

The implementation of the contributions for the BSopt project outlined in this thesis spans over a wide array of related technologies. It includes language-integrated query (LINQ), XPath, C++ interop (IJW), COM interop, Windows Management Instrumentation (WMI), Visual Studio Shell and base technologies such as Windows Forms just to name a few. However the most prominent roles are held by workflows and domain specific languages. The purpose of this chapter is to introduce both on a conceptual level. Additionally, concrete libraries implementing both worfklows and domain specific libraries will be described in further detail.

2.1 Workflows

Businesses have always been trying to formalize and improve their reoccurring processes as more efficient ways to manufacture a good or the removal of bottlenecks helped to advance their corresponding trade. An enabling strategy for this idea was to separate a bigger process into distinct work packages, identify dependencies and the flow of information or objects from the physical world as well as roles and constraining factors. While these processes originally described human interaction, the advent of information technology meant that many tasks such as sending an invoice, doing business related calculations or forwarding documents could be virtualized and integrated into an IT environment thereby relieving employees of any repetitive automatable tasks. As outlined in [12] one way to look at such operations in this context is to separate them into material or information based processes. While material processes are rooted in the physical world and describe the actual assemblage, delivery or some way of handling of physical components, information processes deal with any kind of data suitable for automated processing. Based on this distinction [12] goes on to derive business processes as *"market-centered descriptions of an organization's activities, implemented as information processes and/or material processes"*. In other words business processes are to realize business obligations or to serve customers by the means of suitable information or material processes.

When a company has established to define its business processes it's able to improve those by utilizing the techniques of *Business Process Re-engineering* (BPR). The Workflow Reference Model published by the Workflow Management Coalition (WfMC) [15] defines BPR as being *"concerned with the assessment, analysis, modelling, definition and subsequent operational implementation of the core business processes of an organisation (or other business entity)"*. BPR itself can be supported by the notion of *Workflow Management* (WFM). Its main purpose is to focus on the logistics of business processes and ensuring that at each given point in time the appropriate actions are taken to support the overall business process [1]. The operational perspective of WFM itself is associated with *Workflow Management Systems* supporting this approach. The according definition by the WfMC defines any Workflow Management System as: *Ä system that complete-*

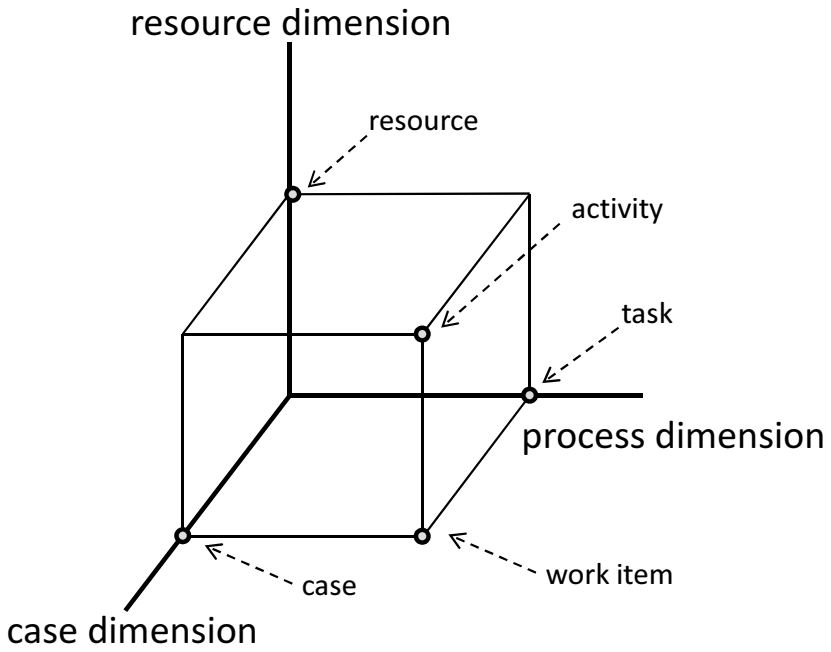

Abbildung 2.1: Three dimensions of a workflow [12]

ly defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic". The term "workflow" itself is rather ambiguous and often used casually. The WfMC defines workflows quite generally as *"The computerised facilitation or automation of a business process, in whole or part"*. For the remainder of this thesis we shall use the definition given by Georgakopoulos and Hornick in [12] where a workflow is *"a collection of tasks organized to accomplish some business process"* with a task being some kind of work to be accomplished by one or more software systems, one or more humans or a combination of those.

Workflows are case-based, meaning that they base their execution on a workflow process definition explicitly tailored for a specific type of operation [1]. This could be the process of document approval in an enterprise, the booking of a flight or the ordering of a product by a customer. Tasks executed in the context of a specific case instance (e.g. ßend seat selection form to customerïn the case "booking of flight 747 for customer Smith") are called "work items" and are mostly linked to a type of resource which may be an automated system such as a printer, a server, a type of robot in manufacturing or a human resource. A work item linked to a specific resource is called an "activity". These definitions serve the purpose for the visualization of a workflow as three-dimensional entity as shown in figure 2.1: here a workflow is depicted as having a case-, a process- and a resource dimension. First all cases are processed independently from each other as visualized by the case dimension. Secondly, the process dimension specifies the tasks and the execution flow. Thirdly, the resource dimension maps to different roles or organizational units within an enterprise. A workflow can thus be represented as a number of entries within this three-dimensional space.

Another way to look at workflows and dissociate the specific notion from the broader context of business process is given in figure 2.2. This illustration is focused on the scope of case-based business processes. According to [1] the specific characteristics of a workflow process are that it is

- case driven

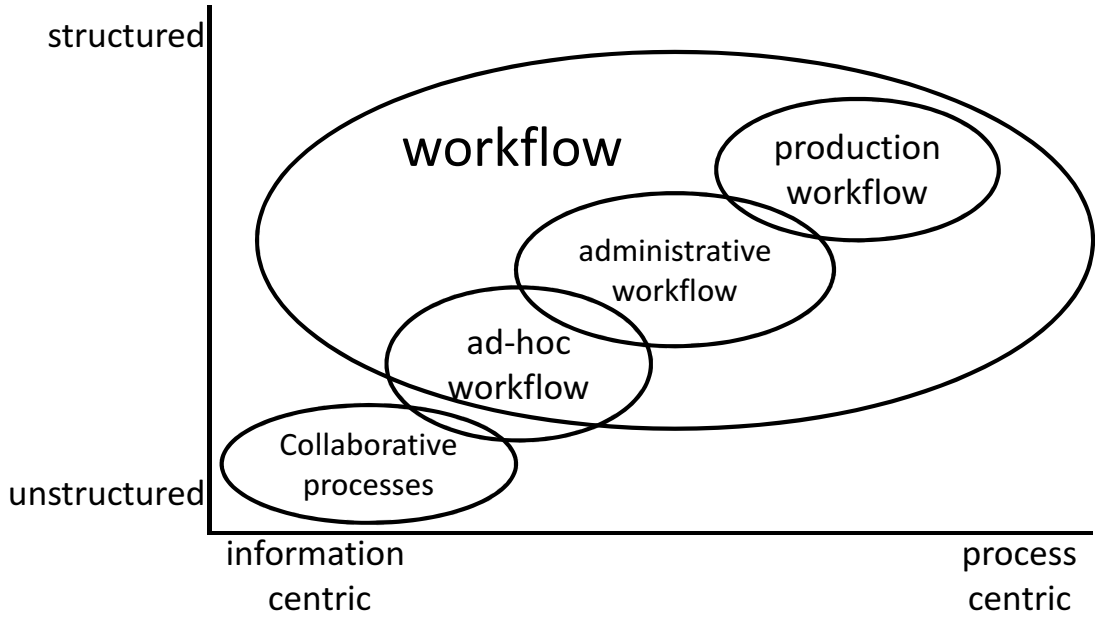

Abbildung 2.2: classification of processes in the context of workflows [1]

- considered to be essential
- explicitly definable in a formal manner

Based on these requirements collaborative processes (not to be confounded with the notion of "business collaborations") are the kind of processes not to be considered workflow processes. This is as they emphasize information sharing and communication in an unstructured way in favor of well defined process centric definitions. On the other side of the spectrum production workflows are very process centric with most cases handled in always the same default way and usually a lot of cases to process. Administrative processes allow for more variation but are still entirely predefined to handle any case possible. At last ad-hoc workflows may be considered as workflows in the above sense but they base their execution on processes not entirely known beforehand. As shown the typification of these processes is not always clear-cut and transitions from one type of process to the next can be fluent but administrative processes and production workflows can still always be defined as workflows.

As illustrated in figure 2.3 the handling of workflows in a workflow mangement system can be separated into three areas: [15]

- build time support to enable the definition of workflow processes.
- run time functionalities to create workflow instances from process definitions and control their corresponding execution flow.
- runtime support for human interaction and the communication with other components in an IT environment.

Build time support is mostly provided in the scope of a separated development environment or system specific development components. It supports the definition of workflow processes by offering workflow specific data structures, commonly required task definitions and means to arrange the overall control flow. This is often supported by visual editing enabling developers to efficiently define processes. [68]

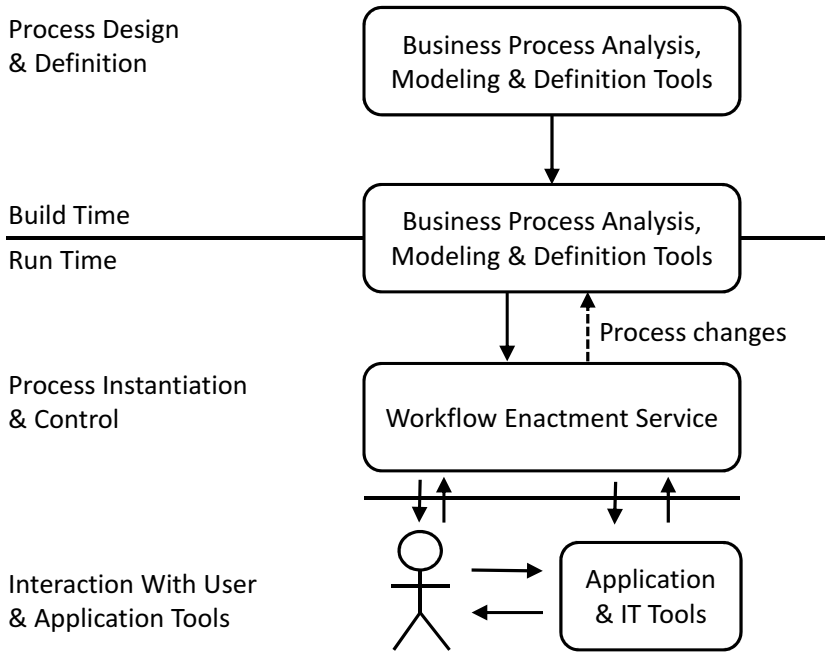

Abbildung 2.3: Workflow Management System Characteristics [15]

The runtime environment of a Workflow Mangement System itself is formalized by the WfMC in the form of a *workflow enactment service* which consists of one or more workflow engines and internal control data. At their most basic form workflow engines are responsible for the instantiation and execution of workflow processes. Zur Muehlen further elaborates in [68] by recognizing eight modules a workflow engine can be responsible for. These modules are able to communicate with each other by utilizing some sort of event handling system as enumerated in the following and summarized in Figure 2.4:

- the *process management facility* takes the role of the entity responsible for instantiation of workflow processes. It must also ensure the validity of any execution constraints specified for the system.
- the *control flow manager*'s task is to oversee state changes in workflow instances and enclosed activities. As part of this requirement it also creates new activity instances to drive workflow execution forward.
- the *worklist handler* manages the interaction between actual participants in a workflow and workitems to be processed in the workflow.
- the *user management facility* handles coordination with system users based on their roles and rights in an organization by utilizing organizational directory services or equally applicable systems (also called "organizational repository").
- the *application invocation module* is responsible for executing external applications, aggregating resulting data and taking notice of any error conditions and return codes provided by these external entities.
- the *data management* component translates data between activity instances.
- the *history management* component is responsible for producing audit data both on a general system level and based on the execution of specific workflow instances.

- the *integration APIs* allow calling applications to access the workflow system programmatically and also provide means to integrate a Workflow Engine into an external system.

With workflows and the idea of workflow mangement systems introduced in this section, the next step is to look at some systems currently in use in the software industry and tested for the BSopt project. The next two sections will look at Microsoft's Windows Workflow Foundation technologies in the versions 3.x and 4.0.

2.2 Microsoft Windows Workflow Foundation 3.x

The Microsoft Windows Workflow Foundation (WF) 3.0 was released to the public in November 2006 and despite its version number is the first product in its line. It is not a deployable product in its own right but is delivered as a library component as part of the Dot Net (.NET) Framework 3.0. This means, that application developers can integrate the library into their own products in order to gain the full benefits of a workflow engine for free. Additionally, a range of products coming from Microsoft also use WF3, most notably the *SharePoint Server* software platform designed for corporate intranets. The .NET Framework itself provides an environment to run 'managed' applications written in languages supporting the Common Language Runtime (CLR) such as C# or Visual Basic .NET and aims to support the trusted and safe execution of code and minimize deployment and versioning conflicts [28]. An update to the WF library raising its version number to 3.5 was added later and brought a tighter integration with the Windows Communication Foundation (WCF) components of the .NET Framework. Workflows can be defined declaratively by creating .XOML files based on the Extensible Application Markup Specification [30] and can include code behind files or be defined completely in code. In order to support this process Microsoft has shipped add-ons for its Visual Studio 2005 and 2008 IDEs so workflows can be described visually as shown in figure 2.5.

The runtime portion of the system is shown in figure 2.6 hosted inside a workflow enabled application process. The process utilizes the functionality provided by WF in order to execute a workflow. The role of the workflow enactment service is represented here by the runtime engine which can host multiple workflows. Additionally multiple runtime engines may coexist in one process. The runtime engine is responsible for instantiating workflows and scheduling their activities by accessing runtime services provided as default implementations or externally added ones. The distinction between tasks, work items and activities is not made in this product. Each entity performing some kind of work, be it at build- or runtime is called activity.

WF comes with a batch of out-of-the-box activities as part of its base activity library. Those include pre-built entities to govern the control flow inside a workflow definition as illustrated in table 2.1 as well as basic multi-purpose elements such as delay-,code- and terminate-activities. Developers are encouraged to develop custom activities when their needs cannot be met by using the basic shipped components. In order to accomplish this task they may derive from given base classes provided by the WF object model. By overriding specific virtual methods and properties new functionality can be provided and harbored in consuming workflow definitions. Alternatively custom activities may also be composed in a visual way. This is supported by a designer canvas included in Visual Studio which allows the composing of new activities out of simpler building blocks. Workflows itself are based on the activities they include. The system provides two default execution styles which are most widely used: sequential and state machine

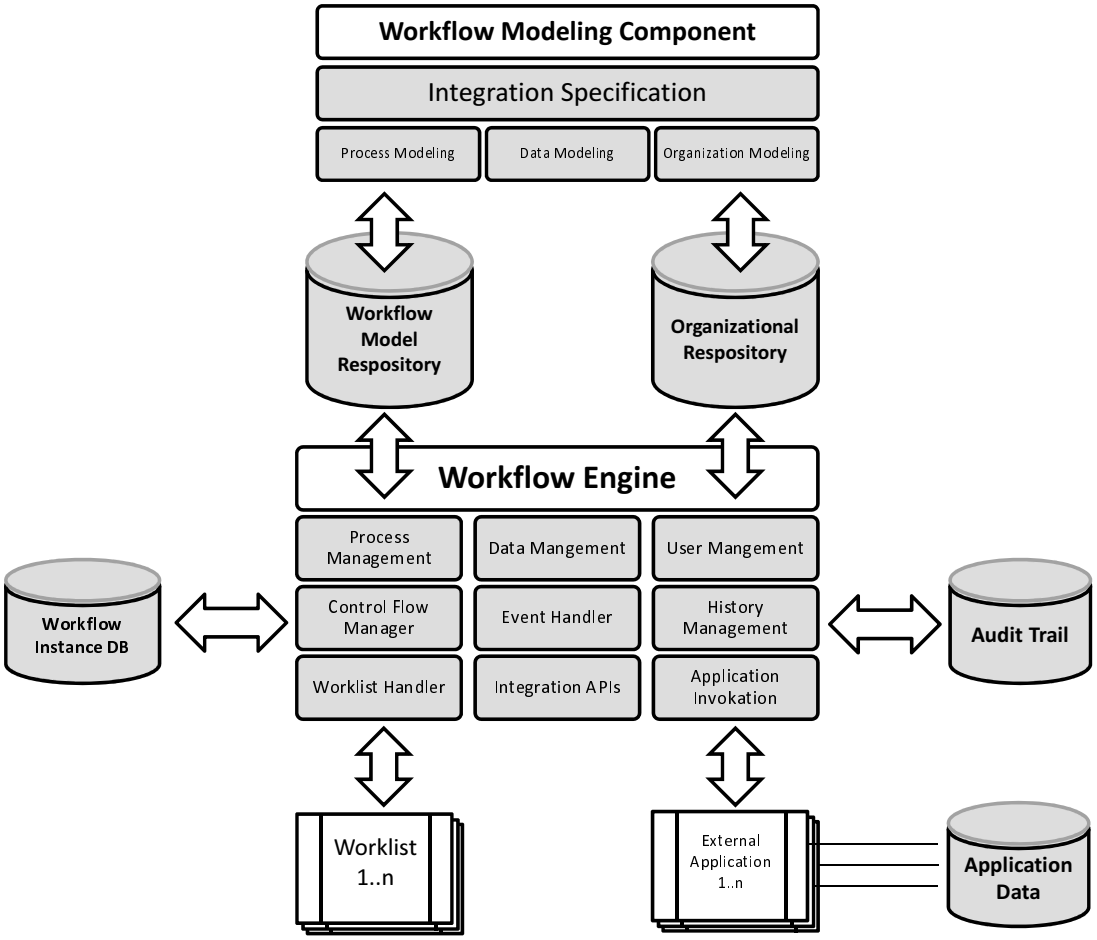

Abbildung 2.4: Composition of a Workflow Engine and its dependencies [68]

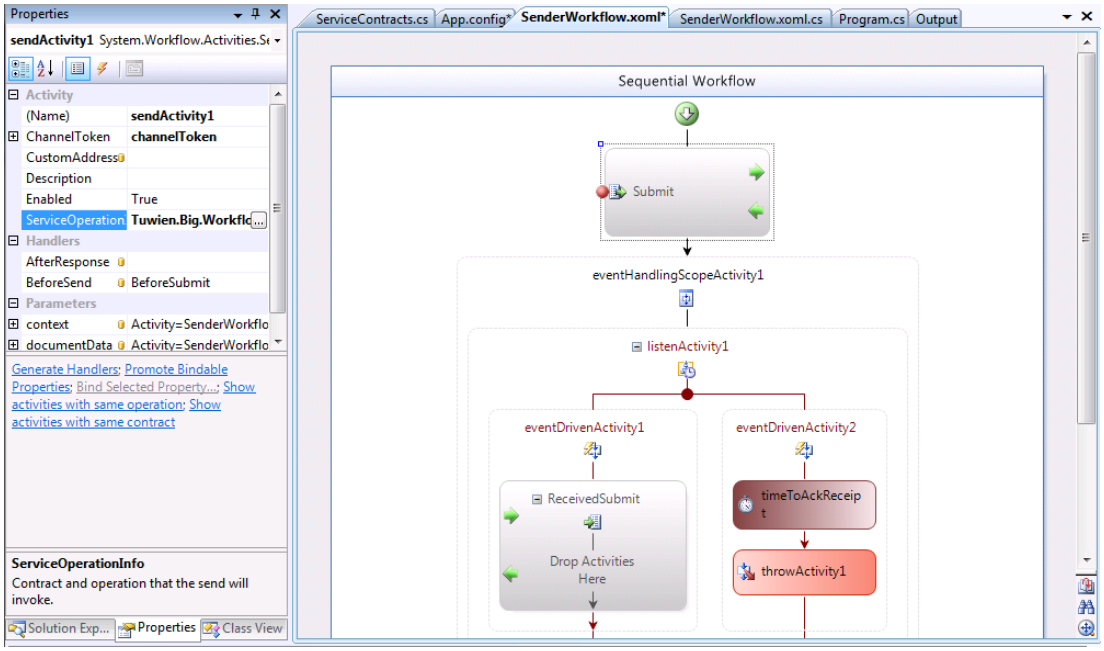

Abbildung 2.5: The WF 3.5 designer showing a sequential workflow definition

based[1]. While sequential execution advances from one activity to the next in a sequence by utilizing special control flow activites (compare table 2.1) state machine workflows define distinct states a workflow can be in and transition conditions to be met for a state change. While the first control style makes it easier to describe and explain what a workflow should be doing it is also less expressive compared to a state machine based approach.

As shown in figure 2.6 WF also comes with a batch of workflow related "services". These are implementations of specific interfaces which the runtime engine can use to accomplish certain tasks. Table 2.2 lists these so-called "base services" which either influence the behavior of the workflow engine or add new functionality to the infrastructure. The workflow "persistency service" is of specific importance in this context as it enables the persisting of an idling workflow instance into a backing store such as a relational database. This allows the system to move workflows between different workflow hosts and take pressure off the host system by relieving memory requirements. Developers are free to add their own custom services to the workflow runtime. These services may either act as replacements of base services or as generic objects used in conjunction with custom activities to support special needs relevant for specific designs. This can be useful for several purposes such as providing a common communication platform for different workflows run by the same runtime engine.

The behavior of activities in WF 3.x can be fine tuned by controlling the properties they expose. There are two kinds of properties: instance properties and metadata properties. The latter can only be changed at design time and in most cases are provided to control the specific operation mode an activity is in. Metadata properties also allow automatic validation of its given design time value. As an example all activities have an Enabled property which can be used to disable an activity e.g. for testing reasons. This can't be changed while a workflow is running. Instance properties on the

[1]The control flow execution semantics of a workflow are in theory freely definable but managing this area is considered an advanced topic not widely pursued.

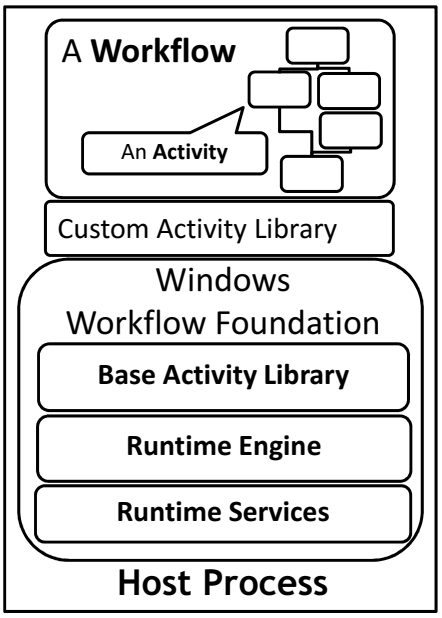

Abbildung 2.6: The hosting of Workflow Foundation inside a process

other hand can be changed at runtime and may either be exposed as normal C# class properties or in the form of so called dependency properties. Dependency properties support binding to data defined outside of an activity which enables flow of information between different activities. The actual definition of dependency properties is rather complicated but the authoring environment provided by Microsoft supports this task well. Listing 2.1 shows the code necessary to define a dependency property of type System.Workflow.ComponentModel.ActivityCondition called ExecutionCondition.

2.2.1 Integration into a service oriented architecture

While the initial release of Workflow Foundation came with certain activities to invoke web services and host services itself it wasn't able to offer a fully configurable infrastructure for communication with heterogeneous systems. This was recognized and solved with version 3.5 which offered a tight integration with the *Windows Communication Foundation (WCF)*, a library about to be presented in this subsection:

WCF is a subsystem included since the .NET Framework 3.0 and allows the creation of highly flexible service oriented applications. It is based on a modular and extensible approach by separating responsibilities into distinct layers and allowing the combination of different layer setups. Figure 2.7 illustrates the architecture of WCF. An application using this framework is free to define its needs on each of the following four areas: [32]

- The "Contracts & Descriptions" area defines how data structures and messages are built up and how the actual message signatures will look like. Additionally bindings are to influence the transport (e.g. HTTP or TCP) and the actual encoding while certain policies can fine tune security requirements.

- The "Service Runtime" layer is to be used at runtime to control certain aspects of the framework such as whether to publish metadata descriptions for services offered, how to deal with error conditions, transactions, the instancing behavior of a service and so on.

Activity	Description
ListenActivity	Enables your workflow to branch conditionally depending on some event or on the expiration of a time-out period.
IfElseActivity	Tests a condition on each branch and performs activities on the first branch for which the condition equals True.
ParallelActivity	Enables your workflow to perform two or more operations independently of each other.
SuspendActivity	Suspends the operation of your workflow to enable intervention in the event of some error condition.
TerminateActivity	Enables you to immediately end the operation of your workflow in the event of some error condition.
WhileActivity	Enables your workflow to loop until a condition is met.
ConditionedActivityGroup	Also known as CAG. Executes child activities based on a condition that applies to the CAG itself, and based on conditions that apply separately to each child activity.
EventDrivenActivity	Wraps another activity and executes it when the specified event occurs.
ReplicatorActivity	Creates and executes multiple instances of a child activity.
SequenceActivity	Runs a set of child activities according to a single defined ordering.

Tabelle 2.1: Control-flow activities from the base activity library [33]

- The "Messaging" layer deals with transport and protocol channels. Transport channels interact by transporting data on the network e.g. over HTTP, TCP, named pipes or MSMQ. Protocol channels are used to augment data with additional headers to facilitate protocols such as WS-Security [50] or WS-Reliability [48].
- The "Activation and hosting" layer supports the hosting and activation of services by different container processes such as IIS, Windows Activation Services (WAS) or self hosting custom executables.

Using and extending WCF is a huge topic by itself. As this section has its focus on Workflow Foundation 3.x only the integration of WCF to support flexible interoperability in a service oriented architecture shall be discussed. As mentioned before, the release of Windows Workflow Foundation 3.5 achieved this goal by providing two new activities: the *SendActivity* and *ReceiveActivity* activities. While *SendActivity* can be used to communicate with an external entity e.g. by calling a SOAP based web service, *ReceiveActivity* allows the implementation of WCF services [5]. In order to call a web service, clients first have to define a service contract interface or let Visual Studio generate this information by deriving it from an already enabled web service which exposes meta data about itself via WSDL [64]. The manual generation of a service contract interface

Service	Description
Scheduling Service	Allows the control of activity scheduling, e.g. synchronously or asynchronously.
WorkflowCommit WorkBatch-Service Services	Allows for fine grained control of the committing process of workflow batches after a transaction, e.g. to introduce specialized error handling.
Persistence Services	Supports the runtime by persisting idle workflow instances to a given data store.
Tracking Services	Comparable to the history management component described in [68]
Workflow Loader Service	Enables the generation of workflow definitions from input formats other than XAML.

Tabelle 2.2: Base Services utilized by the WF 3.x runtime engine

```
public static DependencyProperty ExecutionConditionProperty = DependencyProperty
    .Register("ExecutionCondition", typeof(System.Workflow.ComponentModel.
    ActivityCondition), typeof(ActivityLibrary2.Activity1));

[DesignerSerializationVisibilityAttribute(DesignerSerializationVisibility.
    Visible)]
[BrowsableAttribute(true)]
public System.Workflow.ComponentModel.ActivityCondition ExecutionCondition
{
    get
    {
        return ((System.Workflow.ComponentModel.ActivityCondition)(base.GetValue
            (ActivityLibrary2.Activity1.ExecutionConditionProperty)));
    }
    set    {

        base.SetValue(ActivityLibrary2.Activity1.ExecutionConditionProperty,
            value);
    }
}
```

Listing 2.1: An example Dependency Property definition as described in [33]

consists of defining an interface and the methods it exposes. The interface definition as well as the method definitions must be attributed with meta data in order to further describe each fragment. A sample service contract definition is illustrated in listing 2.2. A send activity in a workflow definition has a *ServiceOperation* property which allows developers to select an operation from all given service contracts and expose any in- or outgoing parameters to the workflow. Receive activities also feature this property which enables them to import a given service contract operation. Additionally, service operations can also be specified in the workflow itself. Workflows can be defined so that their hosting environment automatically creates new instances when a special expected message is being received. For this to work the workflow has to begin with a receive activity with its *CanCreateInstance* property explicitly set to `True`. If an incoming message's signature matches the signature expected by this first activity, the workflow host launches a new workflow instance and hands it the received data. Workflows hosted this way are also called "service workflows". The possibility of a workflow host creating new

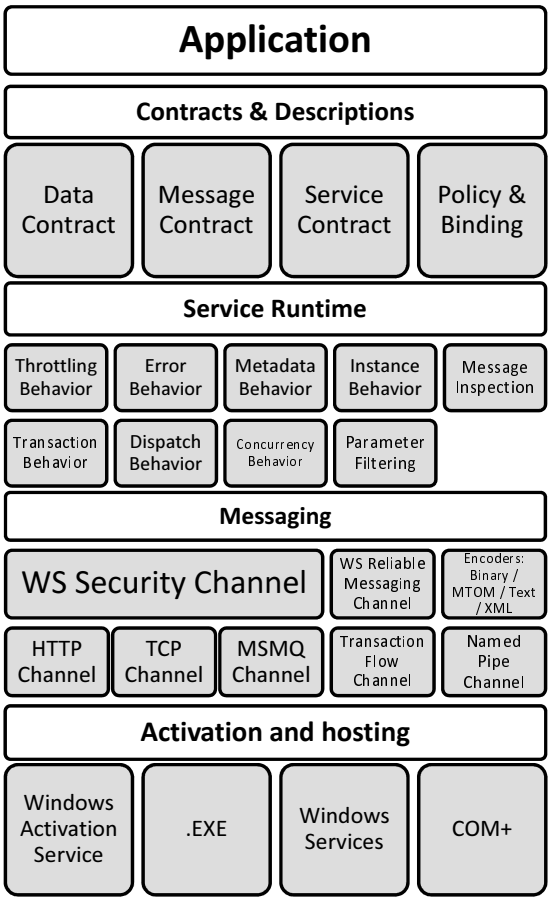

Abbildung 2.7: The WCF 3.x architecture [32]

```
[ServiceContract(SessionMode=SessionMode.Required)]
public interface IBusinessDocument
{
    [OperationContract(IsInitiating = true, IsOneWay = false)]
    void Submit(string documentData, IDictionary<string, string> context);
}
```

Listing 2.2: A sample service contract definition

instances based on specific ingoing messages also hints that the hosting infrastructure implicitly supports hosting multiple workflows in different states at each given point in time. The question arises how an incoming message can be dispatched back to the appropriate workflow instance, an area called "workflow correlation". WF 3.5 supports correlating messages by using *WCF context correlation*, a process described in [46,65]. It is based on the idea of adding a unique identifier to an original message. The identifier can then be used by responders when replying to reach the originally sending workflow instance. This additional meta information enables a workflow host to dispatch incoming messages appropriately. The process is supported by specific WCF bindings which will enable the inclusion of context data via HTTP cookies or as SOAP header as described in detail in the *.NET Context Exchange Protocol Specification* [41]. The preferred way to exchange context tokens when first contacting another party is to make the data an explicit part of an operation. Other options and their pros and cons are discussed in [66].

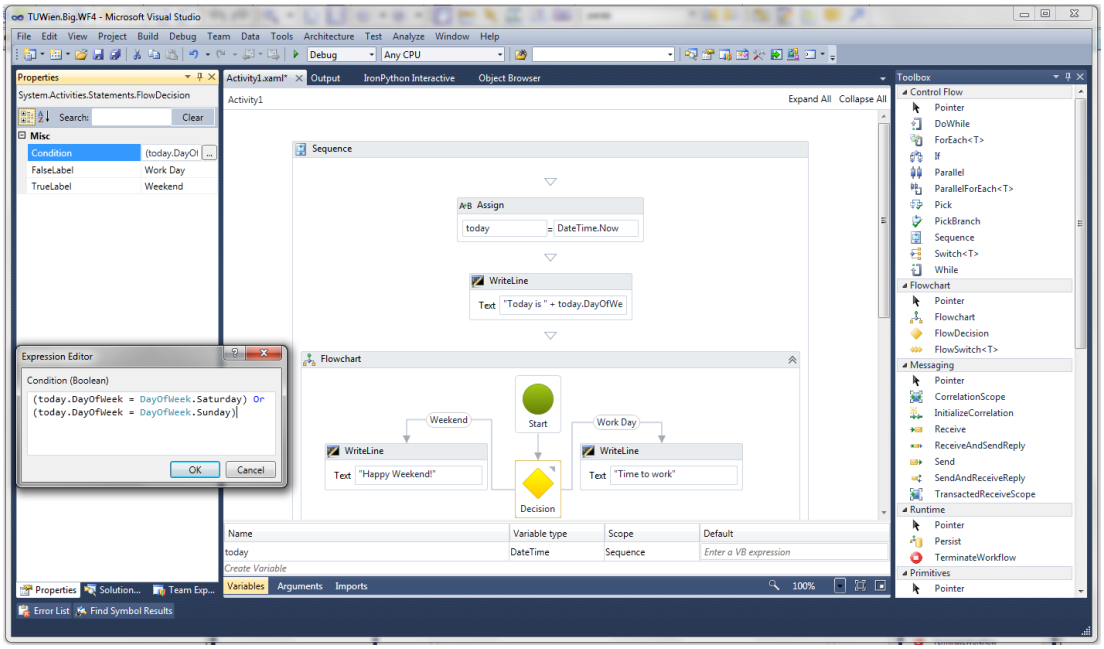

Abbildung 2.8: The WF 4.0 designer with an example workflow

Listing 2.2 shows the definition of a service contract interface called *IBusinessDocument* with one *Submit* operation. The provided context token which is represented as a string dictionary holds the unique id identifying the sending workflow instance. It is explicitly given so the other party can call back using this data for its context binding when necessary. Apart from supporting correlation it shall also be noted that context bindings are useful in defining "durable services". This means that due to the shared context between client and server it's also possible to contact a given workflow instance after a long time, or when a connection has been closed forcibly e.g. due to a server crash. On the downside parties involved in a workflow process must be aware of the contextful information so they can include or prune the information.

2.3 Microsoft Windows Workflow Foundation 4.0

The Windows Workflow Foundation 4.0 (WF4) was released in its final form in April 2010 as part of the .NET Framework 4.0 preceded by a number of public "community technical preview" (CTP) and beta releases. It is a complete rewrite of the workflow system offered before with a number of new concepts introduced. Although it is not compatible with WF 3.x per se it allows developers to include legacy activities wrapped inside an "interop activity". Additionally, the .NET Framework 4.0 still comes with everything needed to develop workflows for the older workflow system.

This section aims not to define WF4 by describing all the differences to its predecessor. Instead the library will be presented in a holistic way which also includes all important features missing in WF3. In [3] WF4 is introduced with the following benefits:

- scalability
- support for persistence
- automatic coordination of parallel work

- automatic tracking
- visualizing of processes

These points of interest will be explored in more detail in the following. Chappell points out that code written in a traditional way, while easy to understand, is hard to make scalable [3]. For instance, business logic waiting for a reply from a web service normally can't just be persisted to a database in order to free resources. Furthermore this also prevents the code from continuing its execution on another computer in order to support a scalable approach based on the anticipated workload. On the other hand, when traditional code is designed for scalability it loses its simplicity as its structure has to be redefined into independent chunks of execution. Furthermore, while the execution sequence of the code was given implicitly before, it's now necessary to explicitly test the validity of the current control flow. This is because state and flow of control get fragmented in this process. The solution, Chappell points out, is to let a workflow runtime handle state management and enable developers to maintain a unified view on the control flow of the business logic. Figure 2.8 illustrates how the control flow defined for a business process can be visualized in a WF4 workflow.

Next he tackles the problem of a process waiting for external input. This may take a long time but still, traditionally the component executing this business logic has to stay in memory consuming resources while just idling. The workflow runtime helps to absorb this impact on available resources by detecting idle workflows and persisting their complete state into an external persistence store. This action frees up any resources the workflow previously consumed. When the runtime detects input the persisted workflow was waiting for, it regenerates the instance from the data store in order to resume the execution of business logic. Another advantage of this approach is that resumed instances may be launched on a completeley different system altogether again supporting scalability.

The synchronization of parallel work can be tricky as race conditions or deadlocks might be the result when handling the process in code. WF4 helps avoiding these problems by providing activities to perform parallel work. Developers are not inclined to manage semaphores or any other synchronization constructs this way as visual composition in the designer environment allows them to put different tasks side by side in a parallel activity to express the simultaneous nature of the described process.

Tracking is another feature which needs much work to be supported in traditional business logic as developers must inject appropriate constructs all over their code. As any workflow instance is transparent in its execution and state to the WF4 runtime, the system allows fine grained introspection and thus supports the activity of tracking for visualization or debugging purposes.

Finally, as in its predecessor, WF4 comes with a visual designer to support workflow creation. The designer is now based on the *Windows Presentation Foundation (WPF)* which helps to visualize complex workflow constructs much quicker than in WF3. Additionally the inherent intent of a process can be displayed in a much more straightforward way by plotting it visually. This is also supported by the new flowchart control flow style. While state machine workflows in WF3 could be hard to explain or introspect visually, flowchart workflows now support a visual style resembling a directed graph including the support for loops. Figure 2.8 shows that different workflow styles also can be mixed as the example showcased in it starts off in a sequential way only to transcend into a flowchart when needed later.

While workflows in WF3 could be based on XAML descriptions the system made it hard to create purely declarative workflow schematics without any additional code. This

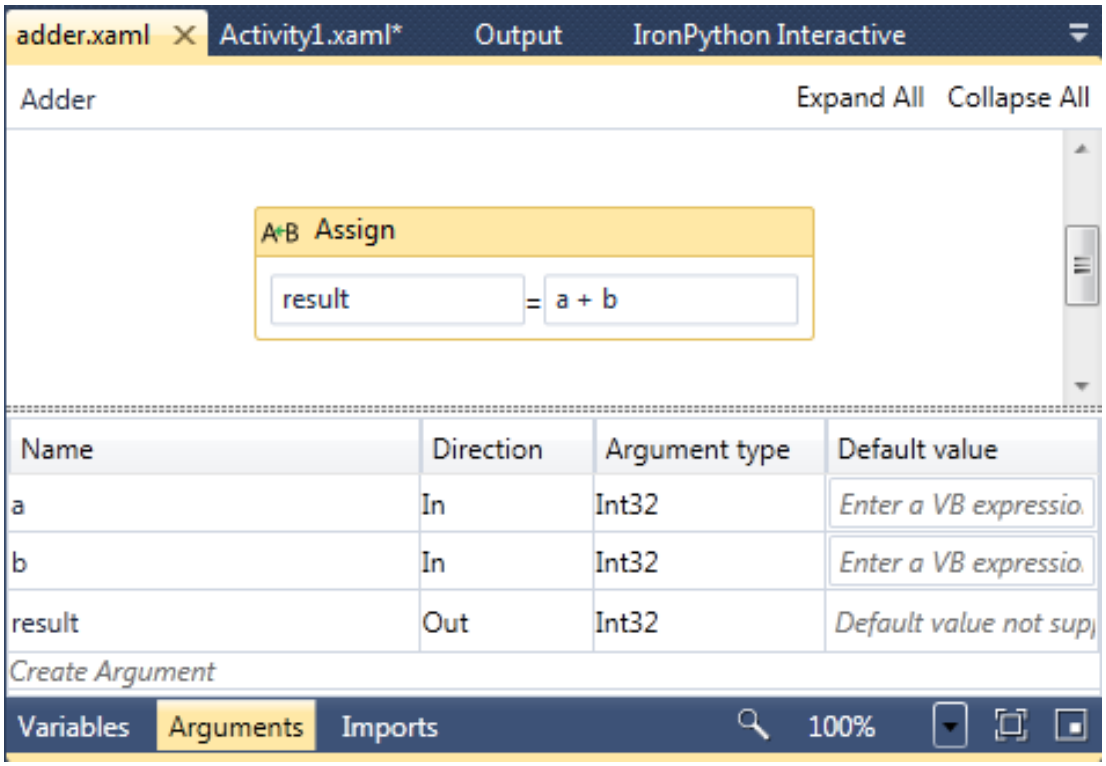

Abbildung 2.9: A simple custom activity for adding numbers

was enforced by the way data was handled by activities. WF3 did not feature a way to define new variables declaratively which meant that in a purely declarative fashion only data exposed by other activities could be bound to. In practice developers had to support this idea by rolling out assemblies with specific custom activities and use their self made infrastructure to define business logic purely in XAML. WF4, building up on a revised XAML specification [42] changes this by integrating scoped variable definitions and data flow into workflow declarations. In fact, when authoring a workflow, code behind files are no longer an option. WF3 activities featured events a developer could hook into in order to react to changed conditions in a code-behind file. WF4 integrates this custom logic by introducing an expression language based on the syntax of Visual Basic. This enables developers to enter their code as inline workflow expression which are no longer separated from the workflow itself. Figure 2.8 shows an expression editor including an evaluation expression to determine whether the current day of the week is either Saturday or Sunday. This expression is used as property definition for the decision activity inside the flowchart which will branch to one of the two possible execution paths.

The concept of dependency properties is not reused in WF4. The flow of data in- and out of activities is explicitly described by so called “arguments”. This strategy transforms activities into entities which take incoming arguments, use them in their business logic and output any outgoing arguments back to the enclosing workflow environment. This concept has similarities to the process of calling a method and thus is a suitable abstraction for developers. Figure 2.9 illustrates this process by showing the visual definition of a custom activity used for adding integers. The ingoing arguments a and b are to be added and stored into the outgoing result argument using an assign activity. When instanced in another workflow or custom activity[2] the system inserts one entity

[2]technically the system does not distinguish between workflows and activities

with a,b and result properties. The direction of arguments is also used in the static validation process of WF4: as the result property is marked as outgoing-only, assigning it a constant integer value would result in the expected error message "Invalid L-value expression".

2.3.1 Integration into a service oriented architecture

WF4 comes with a set of messaging activities based on the updated WCF 4.0 framework. As some concepts in this area have been changed between WF3 and 4, Microsoft has released a services migration document [44] as part of a collection on migration guidance. Some of the most notable news come from areas of service contract specification and correlation.

WF3 Developers either had to manually specify a service contract interface or infer the information from a WSDL description. Receive activities optionally could also embed a service contract specification as part of the workflow. WF4 goes further into that direction by automatically inferring and exposing service contracts from the Receive- and Reply activities used inside a workflow definition. Importing external service contract interfaces is no longer possible but might be added later on in a future release or service pack [3].

While variants on context based correlation summed up by the definition "protocol based correlation" are still possible as in its predecessor, a content based approach has been introduced in this version. This enables developers to let a workflow host dispatch incoming messages back to specific workflow instances based on some kind of content included in the message such as a customer id or other uniquely identifying data structures. Albeit an extensible mechanism, the default way for working with content based correlation is powerful enough for most scenarios. It works by initializing a correlation token by specifying an "XML Path Language" (XPATH) [63] expression identifying the data to be used for subsequent correlations. By referring to this token later on it's possible to correlate messages without using special bindings or making the information part of operation signatures. In order to support this process, a number of custom XPATH functions have been added for this scenario e.g. to help find the start of the actual body in a received SOAP message.

2.3.2 Comparing WF 3.0 and WF 4.0

As previously mentioned, WF 4.0 is a completely independent workflow library not building up on the codebase from WF 3.0. Both systems share a multitude of similarities such as resembling control flow paradigms, a designer aided editing experience, WCF integration and the same service concept. However, an evaluation of the feature set for both libraries concluded, that the requirements of the BSopt project are better met by an implementation based on WF 4.0. The reason for this decision is based on the following unique capabilities of WF 4.0:

- The WF4 design encourages completely declarative workflow definitions. This encompasses the data handling perspective as well as the possibility of using inline expressions to formulate logic. In contrast dynamically defining and linking code-behind files with a workflow definition would be a much harder and less straightforward strategy.

[3] http://social.msdn.microsoft.com/Forums/en-US/wfprerelease/thread/09c40427-a974-4233-ab03-b5bf88c885f2

- The new flowchart control flow allows the direct translation of business process models which base their execution logic on the same concept. At the time of evaluation it was already clear that the source models used by the BSopt team were perfect matches in this context.
- Content based correlation frees the design from a dependency on specific contextful bindings which was used for correlation in WF3. This also simplifies the interoperation with other platforms as data may be transported without extra header information to be considered.
- The automatic deduction of service contracts from a given workflow frees developers from manually defining service contract interfaces. This is especially useful when the workflow definition is created dynamically as no managed source code has to be defined, compiled and linked with a workflow definition on the fly.
- Due to the completely new take on workflows WF3 is a legacy library already. While Microsoft still supports the older system there are reasons for this drastic approach. Customer feedback showed that companies required better performance, easier ways to declaratively define workflows and an overall easier usage experience [56]. Backward compatibility was one sacrifice which was necessary to enable those demands.

2.4 Domain specific languages

Domain specific languages (DSLs) are computer languages *"of limited expressiveness focused on a particular domain"* [7]. Opposed to general purpose languages such as C++ or Java which are designed to instruct a computer to perform arbitrary tasks, DSLs are created with simplicity in mind. This means, that it shall be possible to express all necessary concepts of a domain using the language but not get any more complicated. While limited expressiveness might have an undesirable notion to it, it's actually a big advantage in this case. The less generic a language is the easier it is to use and understand. This makes it a more powerful alternative to using a general purpose language. Fowler identifies four characteristics which must hold to define a language as DSL more concisely [7]:

1. must be a computer programming language.
2. the nature of the language must allow the definition of specific information by a fluent combination of expressions genuine to the language.
3. limited expressiveness with superfluous high level language features and abstraction concepts missing.
4. strict domain focus with well defined constraints.

The notion of DSLs can be separated into three distinct patterns called internal DSLs, external DSLs and language workbenches [7]. External DSLs are used by components in a system to help with accomplishing a domain specific task and are characterized by their differing syntax and semantics. Well known examples for the usage of external DSLs are SQL statements, regular expressions or XML configuration files.

Internal DSLs[4] on the other hand are based on the same language as the remainder of a bigger system is written in but are tailored to only use a limited amount of

[4]also sometimes referred to as "embedded DSLs"

```
private void makeNormal(Customer customer) {
    Order o1 = new Order();
    customer.addOrder(o1);
    OrderLine line1 = new OrderLine(6, Product.find("TAL"));
    o1.addLine(line1);
    OrderLine line2 = new OrderLine(5, Product.find("HPK"));
    o1.addLine(line2);
    OrderLine line3 = new OrderLine(3, Product.find("LGV"));
    o1.addLine(line3);
    line2.setSkippable(true);
    o1.setRush(true);
}
```

Listing 2.3: Example code for setting up a customer in a usual imperative way [8]

```
private void makeFluent(Customer customer) {
    customer.newOrder()
            .with(6, "TAL")
            .with(5, "HPK").skippable()
            .with(3, "LGV")
            .priorityRush();
}
```

Listing 2.4: Listing 2.3 rewritten in a fluent style [8]

available language features. The style the offered operations are named and their return values create a natural way to use the DSL. One example for an internal DSL is the Rails framework for Ruby and more generically internal DSLs are characterized by the "FluentInterface" design pattern [8]. The transformation of listing 2.3 into listing 2.4 shows how it's possible to create an internal DSL by cleverly designing methods aligning with the specified domain's characteristics. Note that the transformed java code is more easily readable by humans and is able to omit the explicit usage of temporary objects created only to aid setting up the customer instance. Another side effect of using internal DSLs is that it still comes with full refactoring and debugging support by the used development environment which is much harder to achieve using external DSLs.

Finally language workbenches are environments to enable "Language Oriented Programming" which uses DSLs as a main strategy to build software [6]. As illustrated in figure 2.10, in contrast to common text based computer languages where the source code as storage format equals the editable component of the model language, DSLs in language workbenches separate storage and editable representation. Users then directly manipulate the abstract representation of the model by working with its projection provided as editor environment which might be completely graphical. Furthermore any processing steps such as generating executables from the model can also be based on the abstract representation which always stays in the background. This approach is contrary to the way traditional source code compilation works where an abstract representation of the input data (e.g. an abstract syntax tree) is only transiently used for object file generation.

While using domain specific languages can be very beneficial, it's also linked to some extra investments in the creation phase. Internal DSLs come with the advantage of an easy automatic integration into any developer tool as they do nothing more than building up on a language already well integrated into a tool chain. This means that features such as automatic expression completion are naturally available. On the other hand the semantic possibilities of internal DSLs are also restricted by the used language which sometimes might not be sufficient enough. Here external DSLs offer much more semantic freedom due to limitless design possibilities inherent to their nature. On the

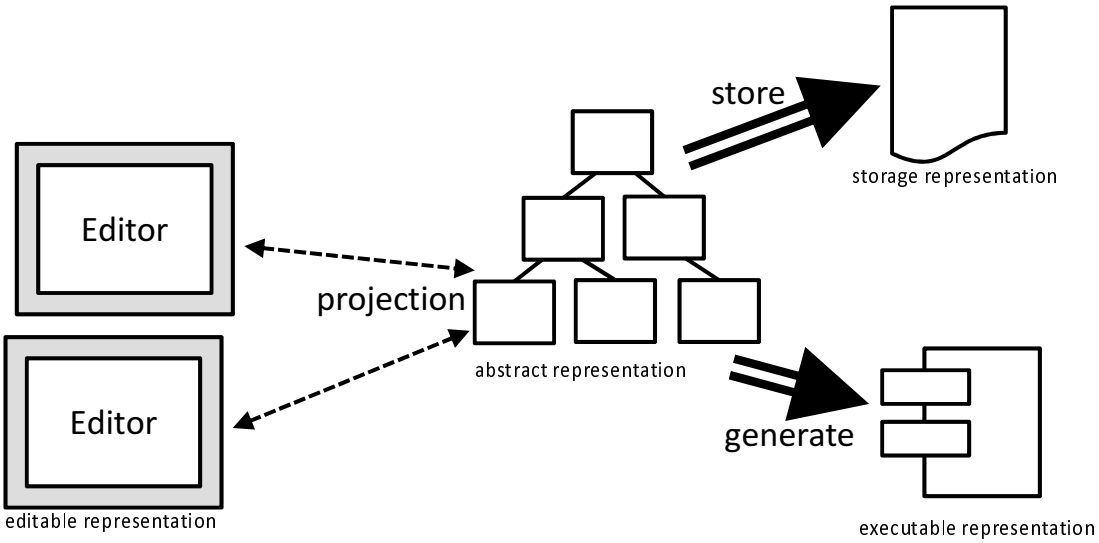

Abbildung 2.10: DSLs inside a language workbench [6]

other hand their integration into a development tool is costly. Language workbenches try to unify the advantages of both types of DSLs just mentioned. The tool integration is designed to support DSLs by allowing the editing of abstract model representations which can be very powerful. Compared to internal and external DSLs, language workbenches are of course more constrained to their editing environment. A standard model for interoperation between langauge workbenches has not been agreed upon as of the time of writing.

2.5 Microsoft Domain-Specific Language Tools

The definition of domain specific languages - depending on its concrete realization - can be a time consuming and error prone process. Microsoft, recognizing the usefulness of this technology, thus began introducing the DSL Tools [26] as part of its Visual Studio 2005 SDK and updated its components for each new version of its development environment. In order to simplify the development of DSLs the modeling environment plugs itself into Visual Studio and is realized as graphical DSL itself. Using this environment, developers may express domain classes and their relationships their target language will consist of in order to form a "domain model" called meta model. Finished meta models can be packaged and used to extend compatible editions of Visual Studio and related shell editions. The technology thus supports the notion of "language workbenches" as discussed in chapter 2.4.

Figure 2.11 shows a conceptual overview of the most important entities used in the domain model creation process. Each domain model consists of domain classes and domain relationships referring to those. A domain relationship itself might either be a reference relationship describing any link between two classes or an embedding relationship where one class is contained in another and subsequently is also destroyed when the parent class is deleted. Both domain classes and domain relationships might hold specific values through the addition of specifically named and typed properties. The domain model and its included class and relationship definitions form the basis for any model instance which bases its structure on this meta information. A model instance must only consist of classes and relationships as defined in its inherent domain

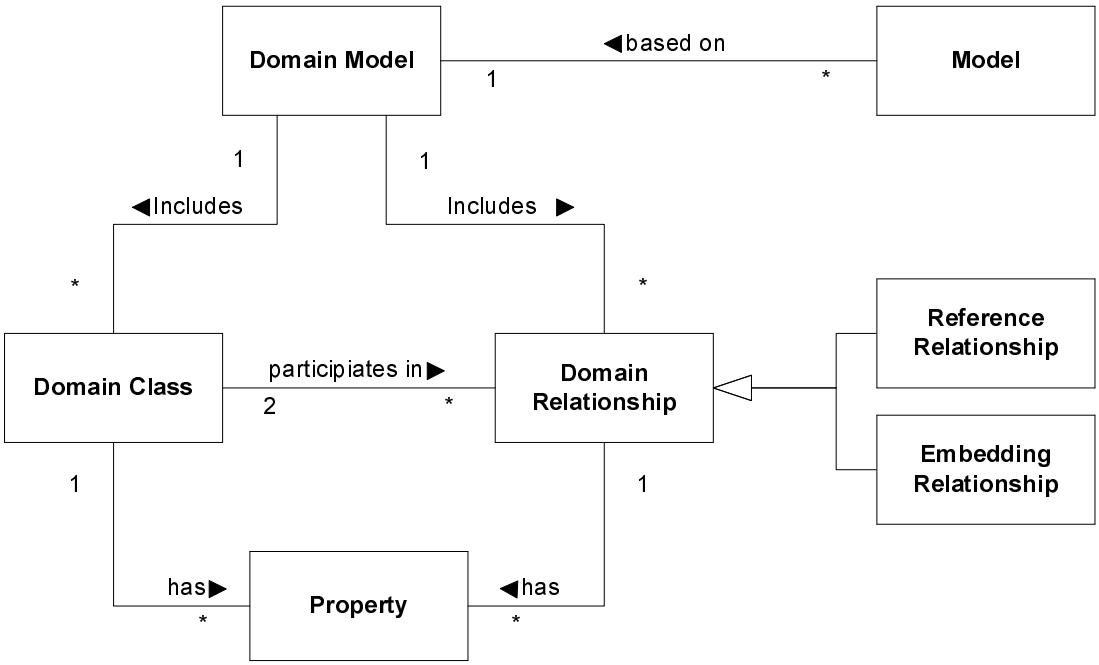

Abbildung 2.11: Conceptual overview of DSL Tools

model. Figure 2.12 illustrates how the meta model editing process looks like by showing a section of the domain model editor from Visual Studio. It pictures the data model of a DSL developed for the BSopt project on the left swim lane. Inheritance is used to define classes deriving from the base class *NamedElement*. It also shows an n:m relationship between *ControlFlowElement* domain classes. The "Diagram Elements" swim lane is used for defining shape elements which are used for visualization in concrete DSL models.

The graphical domain model specification enhances productivity and enables an easier understanding of the DSL's meta model. One thing it cannot offer is the means to define custom validation logic or more generically the fine grained adaption of a domain model through custom code. In order to enable this scenario a domain model offers its own API targeting compatible DSLs. The domain model API itself consists of code transformed from the graphical domain model specification, supporting library code and optional custom code. The abstract representation of a DSL as outlined in figure 2.10 is based on this API and defines the DSL's "domain model" as summarized in [9] as *"An object model of the domain that incorporates both behavior and data."*. In many scenarios DSL models are specifically tailored to support a transformation process. In this way they serve as an intermediate representation on the way to the creation of code or the execution of a specific operation. In principle there are three ways to enable this scenario: (i) text template transformation, (ii) accessing the domain model API and (iii) directly working with the DSL model storage representation.

The DSL Tools use template transformation to create an object model out of the domain model specification defined by users in the DSL Tools editing environment. This technology can also be used for custom model transformation and is especially useful if the desired output is known to have a specific static structure. It is called *text template transformation toolkit (T4)* [34] and is shipped as a part of the DSL Tools. The concept of template transformation is to process a text template containing text blocks and control logic to generate the desired output. In order to enable this

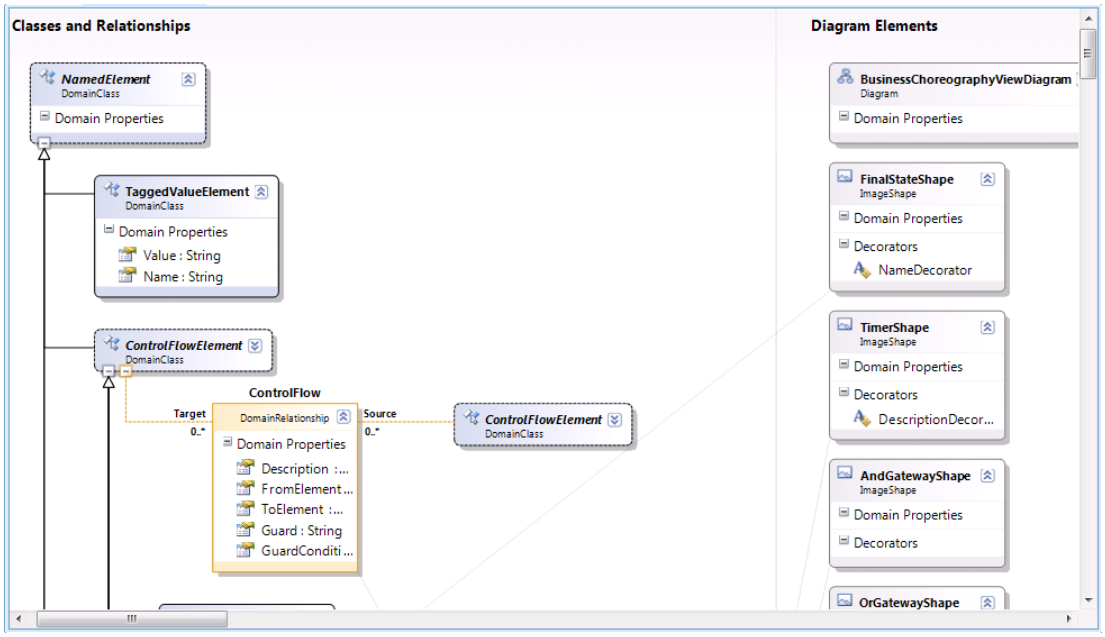

Abbildung 2.12: A section of the DSL Tools domain model editor

scenario T4 uses the services of a given host[5] which acts as base for communication between the user, the external environment and the T4 engine. Directive processors handle special directives in the template by executing type specific code and outputting any results. Default directives for e.g. including files and processing code inline are already predefined. The process of text template transformation can be separated in two steps. First the T4 engine creates a source code representation of the given input. This is accomplished by parsing the data and emitting statements to write any text which is not representing an explicit directive for the system. Built-in and custom directives are processed to create directive dependent output to be inserted into the source code about to be created. Next, this representation is compiled and run in an environment provided by the host. The resulting output is then returned to the host which completes the transformation process e.g. by writing the data into a file [57]. When using this technology in Visual Studio, host developers need to define their text template, add it to a project in solution explorer and run the "TextTemplatingFileGenerator" custom tool on it. As demand in the development community for runtime inclusion of transformation technology surfaced, the release of Visual Studio 2010 came with an extension to this concept called "preprocessed text templates". Instead of the final transformation output, the output of preprocessed text templates is the transformation source code [58]. This can then be used by custom applications to execute transformations at runtime.

The second way to transform a DSL model is by using its API. This also enables developers to create new models from scratch. While model generation in Visual Studio automatically creates layout information by the user putting elements on the canvas, this process has to be imitated by developers generating new models in code. The API supports limited automatic layouting but unfortunately does not support the aligning of nested shapes.

The third option developers have when transforming DSL models is to directly process their storage representation. Models created by the DSL Tools infrastructure are

[5]The default T4 template host is Visual Studio. Microsoft also offers a command line host and custom hosts can be defined by developers.

serialized into an XML based storage representation which is easy to reverse engineer. While it's not as convenient to work with model data on this level, this approach allows to process the model using arbitrary programming languages for XML processing. Moreover this approach also allows the usage of model data on platforms where no dedicated DSL API has been provided.

3 Contributions to the BSopt approach

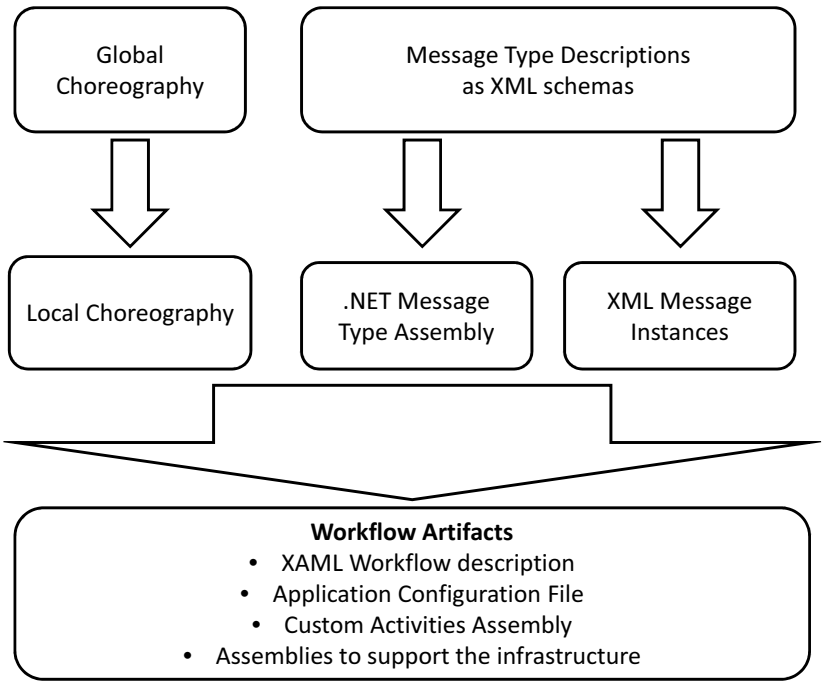

Abbildung 3.1: Concrete steps in the creation process of workflow artifacts as part of the BSopt approach

The main delivery produced by the BSopt project is a development environment supporting an integrated methodology for inter-organizational systems spanning from business models over business process models to their execution in a service-oriented architecture (SOA) [24] (also compare figure 1.1). It acts as a language workbench by providing users with perspective dependent graphical domain specific languages and the means to transform the enclosed information accordingly.

This chapter is about the actual contributions of this thesis for the BSopt approach. The focus lies on the transformation of a given logical business process description into an executable workflow definition. While the handling and transformation of business models in the value perspective as well as the formulation of business documents are part of the BSopt approach, these areas will not be part of further elaborations. However, the interested reader is referred to the website of the BSopt project [24] which includes an ever updated listing of academic papers also dwelling on these topics. Additional information about the BSopt value perspective can also be found in [17, 18, 55].

Figure 3.1 shows the chain of transformations to be discussed in the following sections. It is a substantiated version of figure 1.2 introduced in section 1.2.

In order to generate the WF4 based workflow artifacts shown in the illustration, a business process described from a neutral global perspective, the global choreography,

has to be transformed into a business process description as seen from the perspective of a specific participant - the local choreography. Additionally the electronic messages interchanged in the course of a business collaboration have to be adopted into a format suitable for consumption by the generated workflow artifacts. As will be argued, the message type descriptions, coming in the form of XML schema descriptions, are best transformed into managed types and offered as a compiled .NET class library. Finally, given message type descriptions also need to act as base for the instantiation of actual message instances sent over the wire in order to advance a running B2B process. The final generation of workflow artifacts may be commenced after these three intermediate results have been generated beforehand.

The following sections will both describe BSopt Designer itself as well as the practical contributions for this thesis as outlined in section 1.2. First the DSLs implemented for the realization of global- and local choreographies will be presented, followed by the description of the actual transformation process. Next the handling of message type type descriptions will give way to the concluding description of the workflow artifact generation process. The last section of this chapter will introduce the workflow hosting application both conceptually and from the view of a user. It will conclude the main part of this thesis by demonstrating the actual validity and executability of the artifacts generated in BSopt Designer.

An accompanying example scenario designed to illustrate the actual workflow and further contextualize the ideas outlined in each section will act as final concept validation of the BSopt approach.

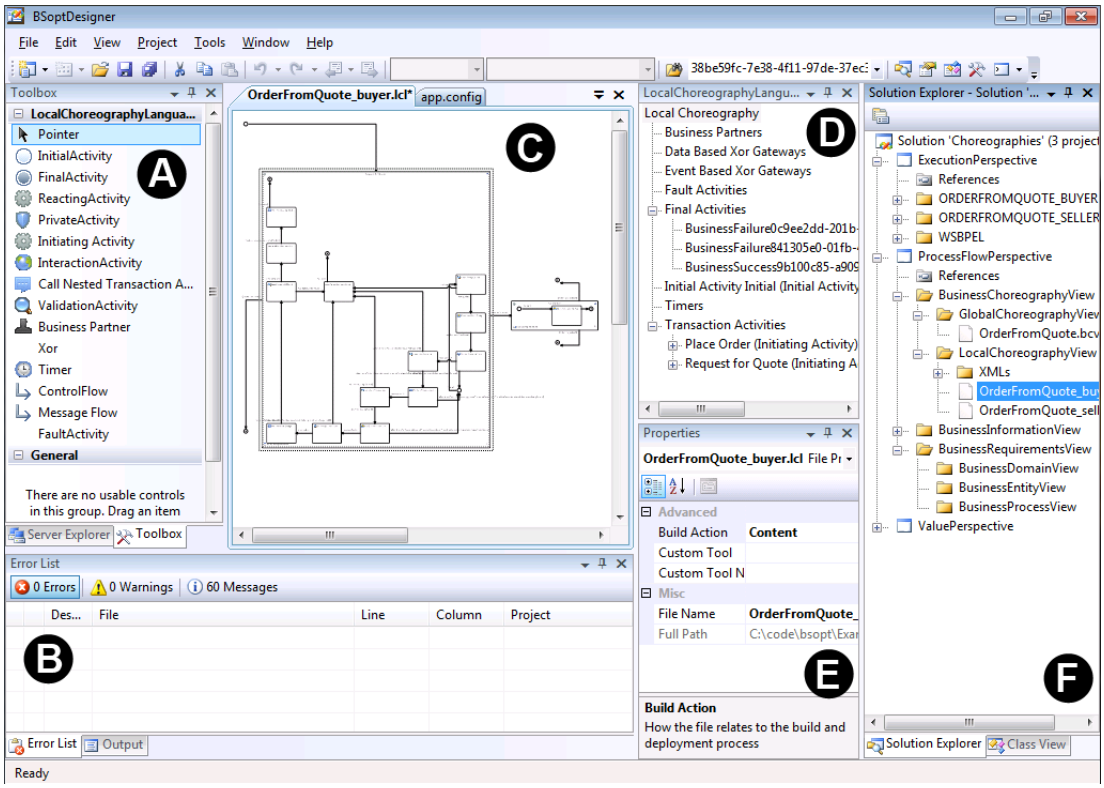

Abbildung 3.2: The primary BSopt Designer user interface components

3.1 BSopt Designer

The BSopt Designer application is a modular tool environment supporting the BSopt approach by providing various graphical DSLs and specialized wizards for data transformation and generation. This section will supply the reader with a general understanding of the vital components of the environment, the ways users are able to interact with it as well as the general tool architecture.

Figure 3.2 shows a screenshot of the application with some example data loaded. The application window is separated into several tool windows which can be repositioned freely by the user and serve distinct but mostly context-sensitive purposes. The handling of data files starts with the solution explorer window (F). Its main responsibility is the visualization of the currently managed files. These files itself are usually linked with a specific kind of project which also may provide specialized context menus or other ways to handle the data. Projects are embedded in a solution which acts as the root of the project management system. Most importantly the solution explorer tool window will provide access to the storage representation of various DSL models. By double clicking an item in the tree view an associated editor will be opened in the editor area (C). An editor is the interactive projectional representation of the item's domain model and allows the manipulation of its data. The Toolbox window (A) is filled with elements related to the current selection context. In the given illustration it offers various constructs to be dragged into the editing canvas for a BSopt Designer DSL. If a component supports the element it will react in a sensible fashion and allow a drag-and-drop event to take place. This way specific data can be added to a DSL diagram or any other destination in the environment. In case the user describes a syntactically correct idea which is flawed at a semantic level such as non allowed circular references

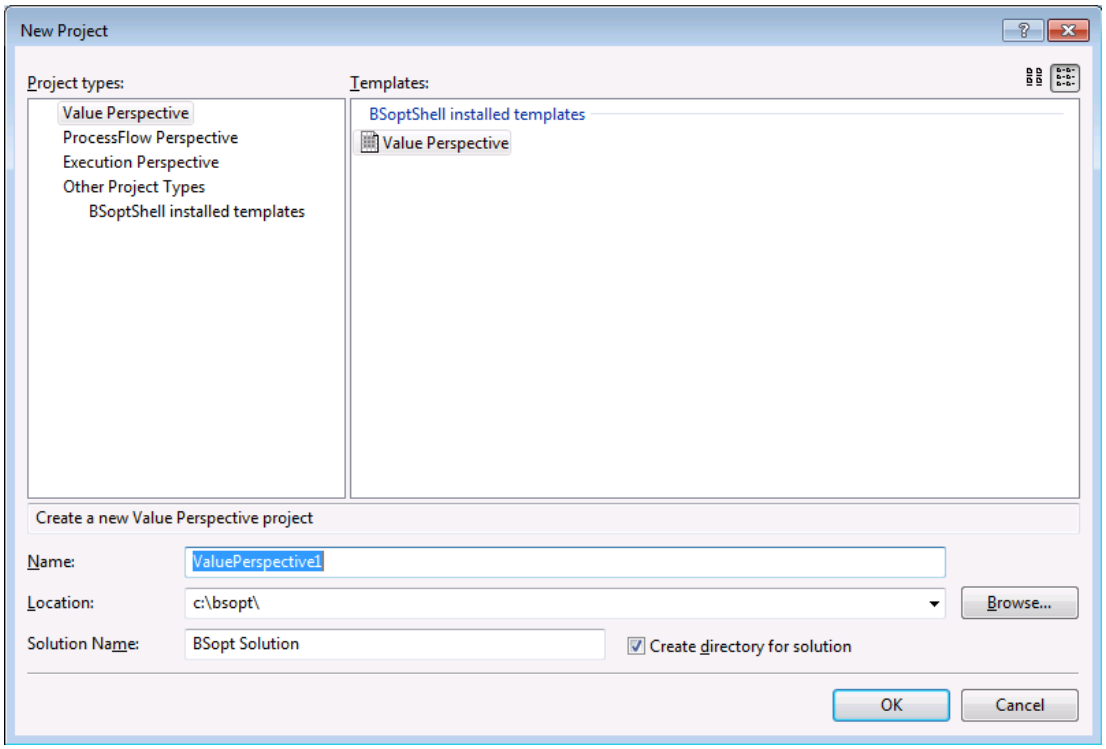

Abbildung 3.3: The BSopt Designer projection creation dialog

it's important to alert the user of the problem. The error list tool window (B) is made for this purpose and more generally for the listing of information, warning and error messages relevant to the current usage context. The output window in the tab next to it is also supporting this task but is better suited for less structured information. Introduced specifically for DSL handling the model explorer tool window (D) acts as a schematically designed representation of the currently active DSL model. It illustrates the diagram in a tree-like fashion and supports the selection of entities in the model based on their logical relation to other elements. Last but not least, the property tool window (E) acts a general purpose utility to visualize any important data related to the current selection inside BSopt Designer. It consists of two columns with the first showing each property's name and the second the according property's value. The value field may be a simple textual representation but can also be a complex designer to help with data entry such as a visual calendar. Properties need not always be editable but most elements inside a DSL model offer their specific properties so that users can adapt the given information accordingly.

With these principal user interface elements explained the next step is to describe how users may start interacting with the application. In order to edit any DSL model it first must be added as part of a project to an open solution. As pictured in figure 3.3 the "New Project" dialog offers project types for each of the three distinct perspectives identified in the BSopt approach: value-, process flow- and execution perspective. When creating a new project the enclosing solution is automatically generated as well. Alternatively a project type for blank solution creation is also offered by the system.

When opened inside BSopt Designer, each distinct project type offers its own special set of item types to add. Figure 3.4 shows the available item types to be added to a "ProcessFlow Perspective" project. As mentioned before double clicking an item will open an associated editor to enable the editing of data. A more thorough examination

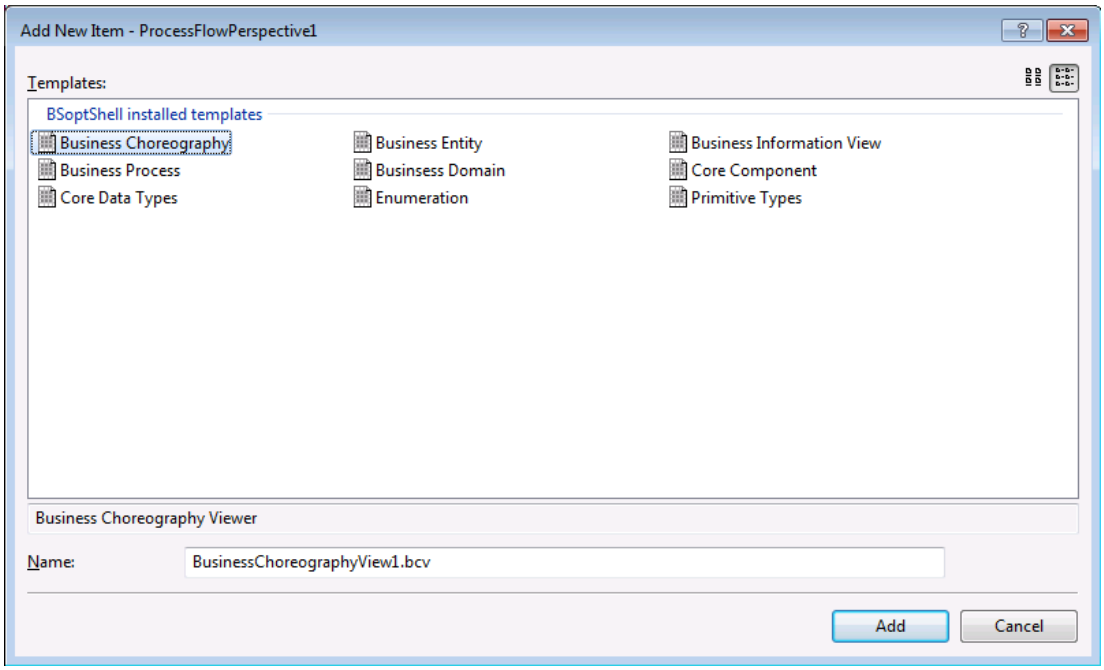

Abbildung 3.4: The Add New Item Dialog for "ProcessFlow Perspective" projects

of DSL editors relevant to this thesis will be presented when suitable in the following chapters. Additionally items in solution explorer may be right clicked to launch a context menu displaying all the possible actions for the item. This context menu approach is used throughout BSopt Designer to offer users the transformation of DSL models by subsequently launching custom transformation wizards.

3.1.1 Architecture

This section will look at the general architecture of BSopt Designer. The application is designed as a Microsoft Windows program mainly developed with the C# 3.0 language [13]. C# is a "managed" language meaning it builds upon the Common Language Runtime (CLR) of the .NET Framework [27]. BSopt Designer is also depending on the .NET Framework 3.5 release which corresponds with C# 3.0. The application further builds upon the Visual Studio 2008 extensibility model. This development environment offers three basic ways for extension [36]:

- The *macro* mechanism can be used to record repetitive tasks and is offered to replay a series of commands. Macros can also be created, edited and debugged themselves which is supported by Visual Basic acting as macro language. While useful in automating smaller tasks, macros are not suited to creation of tool windows or more elaborate tasks.

- *Addins and Wizards* are Component Object Model (COM) based binaries implementing specific interfaces supported by the Visual Studio core automation model. Using this approach it's possible to seamlessly integrate functionality into the environment which supports custom menu- or toolbars this way. Additionally new tool windows and option property pages can be created. Wizards are a special type of addin implementing an interface to support leading the user through a series of steps in order to accomplish a task.

- The *Visual Studio SDK* offers the most comprehensive ways to extend the VS IDE. It is used for integration of new programming languages, project types, editors or debuggers. It offers an object model for native (concrete platform targeting) and managed code based on the concept of so called *packages*. Registered packages are automatically loaded by the IDE and are provided with data necessary for extension. Building on this concept starting with the Visual Studio 2008 SDK Microsoft also provides the *Visual Studio Shell*. Shell instances are based on a series of core packages which provide the basic functionality of Visual Studio without any programming languages and are offered for developers aiming to create extensive solutions for specific domains. Visual Studio Shell solutions come in either integrated- or isolated mode. The former requires a fitting instance of Visual Studio already installed on a system and will integrate itself into the application. The latter, coming with an own "application id", will be provided as a detached application not sporting any further dependencies on a Visual Studio installation at all.

Figure 3.5 shows the building blocks of the BSopt Designer architecture. We chose to use the Visual Studio isolated shell as basis for the application so it can be deployed in environments where no previous Visual Studio versions have been installed. Fortunately using the shell this way is a royalty free process. As can be seen the extension of the isolated shell just described is realized by the addition of a number of domain specific languages. As described in section 2.5 the DSL Tools shipped as part of the Visual Studio 2008 SDK are used to define each custom language finally offered by the tool. The template created in this process also offers a Visual Studio package implementation which, when referenced by the shell, will automatically embed the DSL into the environment. What's not done in this process is to automatically offer project templates for each usage perspective and the accompanying item templates as was shown in figures 3.3 and 3.4. The *Visual Studio Managed Package Framework for Projects (MPF-Proj)* [53] helps in this respect by providing source code for realizing these tasks and was subsequently used for their implementation.

As the overall tool concept builds on the idea of representation transformation it became clear that this process had to be supported by some kind of wizard user interface. Certain items in a BSopt Designer solution offer context menu items to launch these wizards. While the Visual Studio addin model supports the creation of wizards, their design time experience is lacking in comparison to today's standards. The *simple wizard control* [22] introduced as open source library on the developer website "The Code Project" was thus chosen for its easy handling and powerful design time experience as shown in figure 3.6. Developers using the control are able to visually define each individual step of a wizard, drop controls on the according canvas and define events raised when the state of the control is changing.

One workflow system targeted by BSopt is the Windows Workflow Foundation 4.0 (WF4) which ships with the .NET Framework 4.0. In order to create WF4 based artifacts from BSopt Designer, the tool itself must also inherit a dependency on the .NET Framework 4.0. However Visual Studio 2008 based code such as BSopt Designer itself is not able to directly consume code written for the new *Common Language Runtime (CLR)* version introduced with .NET 4.0. As a consequence the functionality used to create and host WF4 workflows has been externalized into separate console based applications which then can be called by the appropriate components in the tool environment. This also bears the advantage that these actions can also be performed independently from BSopt Designer if necessary.

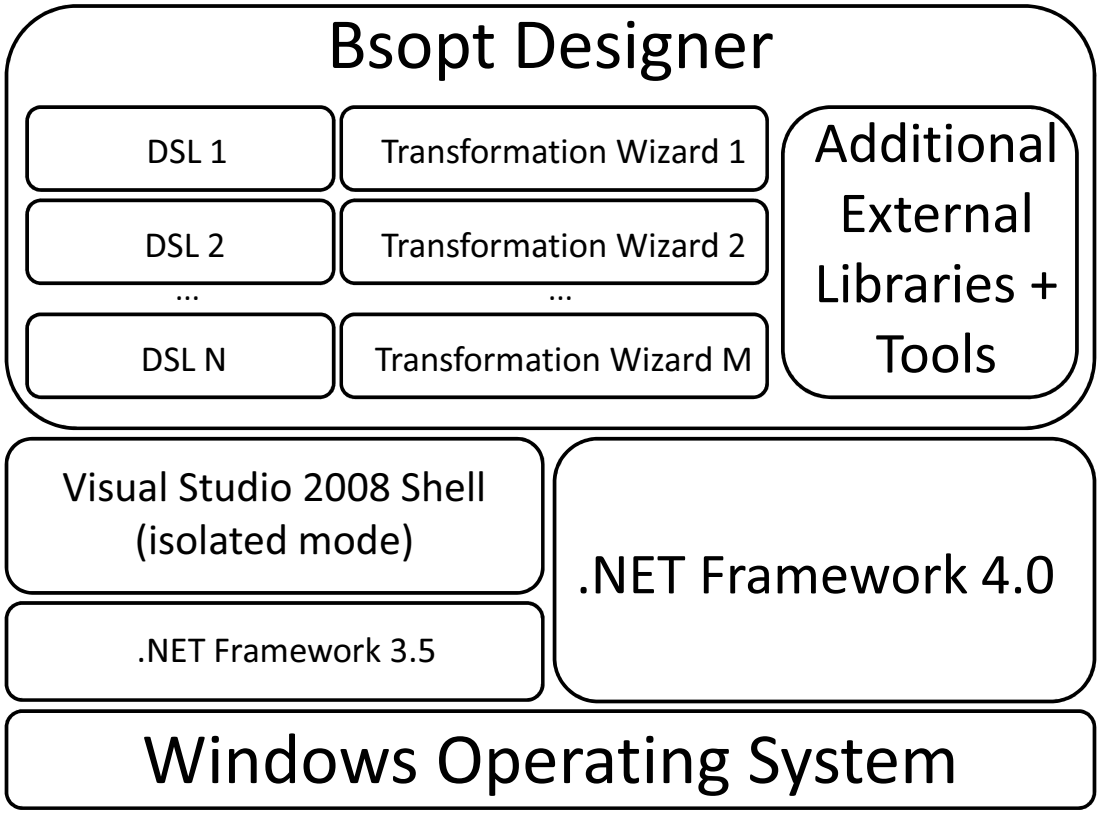

Abbildung 3.5: Building blocks of the BSopt Designer architecture

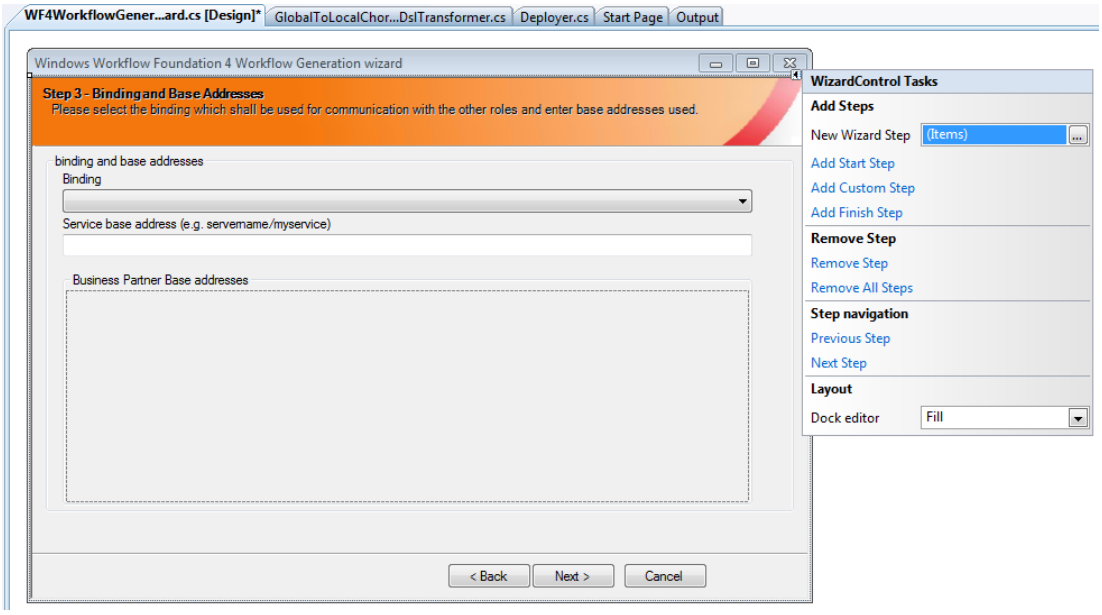

Abbildung 3.6: Design time experience of the simple wizard control [22]

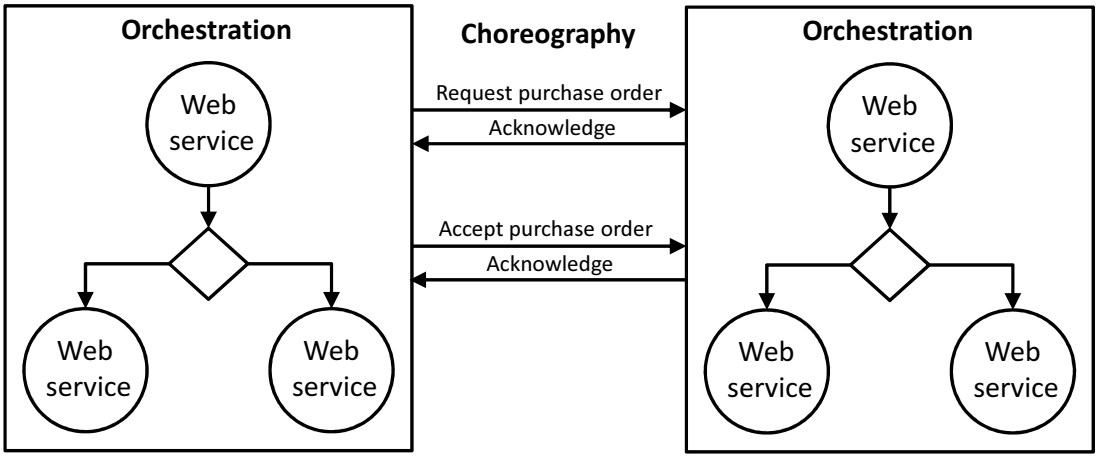

Abbildung 3.7: The differing scopes of orchestration and choreography [54]

3.2 From global to local choreographies

When designing a business process including two or more interacting participants there are two basic ways to start: first the process can be created in a top-down approach starting with a sequence of inter-organizational messages, their contents and the formulation of generally expected behavior. Next the internal control logic and other intra-organizational information may be added by each company involved in the creation process independently. Secondly, businesses may start with the details and end with the big picture of the process at the end. Unless businesses are influential enough to get other participants in the market to adopt their design it's very unlikely for two enterprises to connect using the second approach. This section deals with the first strategy and before introducing concrete implementation details in BSopt Designer it's useful to discuss some terms referred to later on.

The domain of business process modeling distinguishes between the two related concepts of *orchestration* and *choreography*. According to [54] an *orchestration* encompasses the interaction of one party with internal and external web services, the necessary business logic and sequence of actions in order to complete a business process. *Choreographies* on the other hand restrict themselves to the observable sequence of public message exchange between business partners as illustrated in figure 3.7. In [14] Hofreiter further distinguishes between *local-* and *global choreographies*. While *global choreographies* are based on a neutral perspective, *local choreographies* can be seen as projections on a corresponding *orchestration* illustrating only the external message exchange from the point of view of one business partner.

As the difference between the two kinds of choreographies is only their perspective it's always possible to transform one representation into the other. This also means that conforming orchestration skeletons may very well be derived from a global choreography by transforming its representation into local choreographies for all participants. The additional internal business logic described by an orchestration can then be added by each business partner independently without breaking the common contract defined by the global choreography.

The process discussed in this section relates to figure 3.1 and defines one aspect necessary for workflow artifact creation in BSopt Designer. Here it also covers the first major contribution for this thesis, namely the transformation of a DSL model describing a global choreography into a local choreography for each original participant. In order

to properly outline this process it's necessary to review source and target DSLs which will be discussed in the next two subsections.

3.2.1 The business choreography language (BCL)

The motivation for the BCL [47] was to enable the modeling of global choreographies based on the UN/CEFACT Modeling Methodology (UMM) as presented in [16]. While a UML profile for modeling UMM constructs already existed, feedback from real world projects showed that some concepts were hard or cumbersome to express. These problems were identified to result from the need to adhere to the UML meta model constraining the ways to represent global business processes. The BCL aims to replace this modeling approach by embracing the strengths of domain specific languages and allow users to express global choreographies in a more straightforward way while still conforming to the UMM. Altogether [47] identified eight concepts which were translated from their UMM representation into BCL opponents which are briefly discussed in the following:

- a *business transaction* defines the message exchange between exactly two business partners and is the basic component in a global choreography defined by the UMM. Its aim is to update and synchronize both partners' states of information by utilizing uni- or bidirectional message exchange patterns. The legal status is governed by the *business transaction pattern* used for the *business transaction*.

- *business transaction patterns* in UMM follow the definition of the six Open-edi patterns. They may either be unidirectional (notification, information distribution) or bidirectional (request/response, query/response, request/confirm, commercial transaction) and specify the way two business partners exchange business documents. As specified in the implementation draft for UMM v2.0 [16] each *business transaction pattern* comes with a set of default values associated with its *business transaction* which governs its *quality of service* aspects. Additionally, the type of a *business transaction pattern* also directs whether acknowledgment messages are to be expected and for which occasions these will be sent.

- the *quality of service* concept defines security- and communication aspects for a *business transaction*. Those include data such as time frames in which participants need to respond, whether non repudiation of received messages is required and how often senders shall retry to send a message to their business partners before giving up.

- *business documents* are exchanged by business partners in a *business transaction*. In the case of a unidirectional *business transaction pattern* the initiating party is the only one to send a *business document* in order to synchronize the information level of both parties. An example for such a scenario would be the notification of shipment of an ordered item. When bidirectional *business transaction patterns* are used the responding partner will answer with a *business document* defining the final outcome of the business transaction. This translates to a new common understanding of the business situation on both ends. As an example for this scenario the ordering of an item is answered by the item's seller by either accepting or denying a buyer's request.

- a *business collaboration* is a long running business process which enables two or more participants to engage in one ore more business transactions in order to

accomplish their immediate business goals. The business collaboration not only defines the number and specific kind of business transactions but also the control flow between those. It also supports splitting and merging control flow constructs (AND, OR, XOR) so more complicated control routings can be expressed.

- as already mentioned *shared states* for both business partners have to be synchronized accordingly in order to enable participants to properly engage in a *business collaboration*. The *shared state* reached after the completion of a *business transaction* is further important as it influences the continuing control flow of the *business collaboration*.

- *Reuse of business transactions* is a concept to minimize unnecessary duplication of modeling efforts.

- *Role mapping* supports the usage of *business transactions* on an abstract level by referring to roles instead of concrete business partners. This concept is used to map two specific participants in a business collaboration to the initiant- and responder-roles inside a business transaction. Additionally, it also increases the usefulness of *business collaborations* as they can be nested inside other *business collaborations* with certain participants mapped to specific roles.

While not directly derived from the UMM specification, ideas from the Business Process Modeling Notation (BPMN) [52] further influenced the following three concepts which were also introduced in the BCL to extend its expressiveness:

- *timer events* introduce a temporal component into the model by enabling absolute or relative delays of the control flow. This may be used for the modeling of time-out scenarios which could e.g. end a business collaboration after a specified amount of time has passed.

- *compensations* provide a solution for scenarios where process-wide transactions might not be feasible. Long running business collaborations are very applicable to this scenario for resource reasons and thus are well suited for this technology. When participants in a business collaboration experience technological faults forcing them to stop, the process compensations make sure that the already agreed upon state changes coming from successfully finished *business transactions* can be rolled back. This is accomplished by redirecting the control flow toward compensation *business transactions* introduced to restore the state of information before the business collaboration has started.

- *event based XOR split nodes* as presented in [62] provide an opportunity to introduce deferred choices into the model by basing the branching of the control flow on external events such as the (first) incoming message applicable to the current state in the *business collaboration*.

Those concepts just presented are best illustrated by an example DSL model based on the BCL. The following subsection will introduce this example as a starting point of an ongoing process leading to actual workflow artifacts ready to be hosted by a business application.

3.2.2 Introduction of the accompanying example scenario

Before discussing the distinct elements used in the composition of BCL models the example scenario is presented in a more general way: the principal premise driving

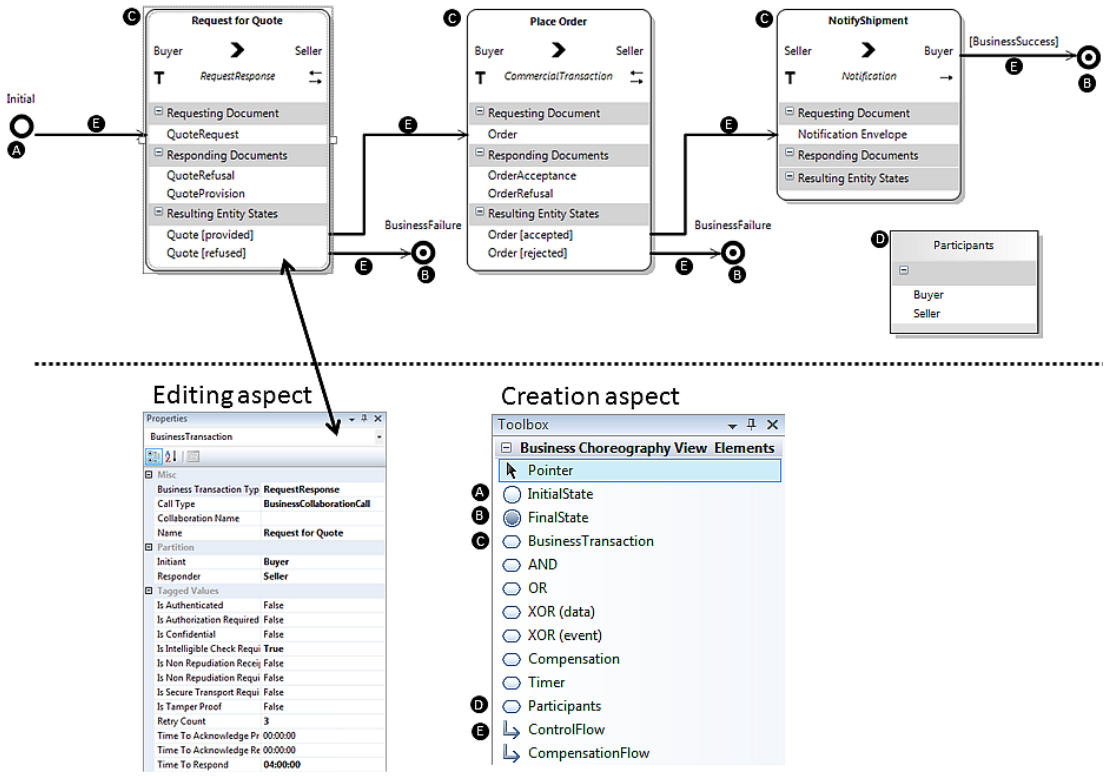

Abbildung 3.8: The "order from quote" example scenario expressed in the BCL

this business process is the interaction between two participants acting as buyer and seller of goods. The general outline schedules the buying party to initiate the process by requesting non-binding quote information for a product from the seller. The seller may or may not comply with this request leading to the common understanding that either a quote has been refused or provided. In case of the latter the buyer continues to place a legally binding order for the quoted item type which is defined to conclude with the seller accepting or rejecting the order. An accepted order will at some point lead to a shipment for the buyer. This will induce a notification for the buyer when the ordered items have been shipped. After this shipment notification has been sent from seller to buyer the business collaboration ends successfully. In case the order was rejected beforehand it ends in a failure.

Figure 3.8 showcases the just described example scenario modeled in the business choreography language. Users of BSopt Designer can create this example from scratch by adding a new file into a processflow project and dropping applicable elements onto the provided DSL editing canvas.

The upper half in figure 3.8 depicts the definition of a *business collaboration* containing three *business transactions*, including two participants acting as buyer and seller. The first *business transaction* follows the bidirectional *request/response* pattern while the second is defined to indicate a two way *commercial transaction*. Finally, the last one follows a one way *notification* pattern. According to the UMM v2.0 foundation module specification draft [16] the *request/response* pattern is used to return *"information that needs to be dynamically assembled and hence cannot be returned immediately"*. In contrast to the *commercial transaction* pattern the involved parties have no residual obligation to fulfill the terms of a contract. Finally, the *notification* pattern is used to inform the other party *"about an irreversible business state"* which is - despite its

informational one way character - verified by the inclusion of an acknowledgment of receipt message as is also done with the *commercial transaction pattern*. Additionally the *commercial transaction* pattern also verifies the successful processing of a sent business document by requiring an appropriate acknowledgment message.

Each element in this area has been dragged from the toolbox window shown in the figure onto the editor canvas where it manifested its shape as defined by the DSL author. The illustration distinguishes those different entities by marking them each with a letter between A and E in order to highlight this fact. Each entity possesses type specific properties which may be changed by the user. The illustration shows this editing aspect by introducing the properties reflected by the "request for quote" *business transaction* as shown in the properties tool window from the editing environment. Those include the type of *business transaction pattern* used, *quality of service* parameters as defined in the "tagged values" section and the mapping of the given participant roles buyer and seller from the *business collaboration* to *business transaction* specific initiant- and responder roles. Additionally, each *business transaction* contains separated compartments defining both the one requesting and all possible (none to many) responding *business documents* which shall be valid in the scope of the *business transaction*. As an example for the "request for quote" *business transaction* this translates to a *quote request* being answered by either a *quote refusal* or *quote provision business document*. Finally, the "resulting entity states" compartment abstracts the new common understanding of both business partners after the completion of the *business transaction*. While the *shared state* of both participants includes the concrete information exchanged using *business documents*, resulting entity states allow to partition these realities into a discrete number of possible outcomes. Control flow entities which are defining all valid routes for a *business collaboration* can refer to these resulting entity states as starting points. Generally and as depicted in figure 3.8 a *business collaboration* includes exactly one "initial state" entity acting as starting point for a business collaboration and one to many "final state" entities ending with either a business success or failure. All non-flow elements in the BCL, but the "participants" entity, must be connected by either control flow or compensation flow shapes in order to be part of the *business collaboration*. The "participants" entity defines all roles in the scope of the *business collaboration* and is only allowed once per diagram. While join/merge nodes and the additional concepts derived from BPMN are also supported in BCL they are not included in this example as they won't be processed by the current transformation engine. Nevertheless they are discussed at length in [47].

3.2.3 The local choreography language (LCL)

Global choreographies define business processes as seen from an external observer's neutral perspective by outlining the types of messages exchanged as well as the sequence of their appearance. As discussed in the previous section, the business choreography language is used in BSopt Designer to support the modeling of global choreographies. Local choreographies differ from global ones in that they show the public message exchange as perceived from the perspective of a participant directly involved in the process. Opposed to global choreographies workflows are generally business process descriptions based on one specific participant's perspective and thus have more similarities to local choreographies and orchestrations. In order to gradually lead the transformation of global choreographies toward workflow artifacts it was deemed sensible to introduce another representation of the business process based on the qualities of local choreographies. The result is reflected in the domain specific *local choreography language (LCL)* which will be presented in the following.

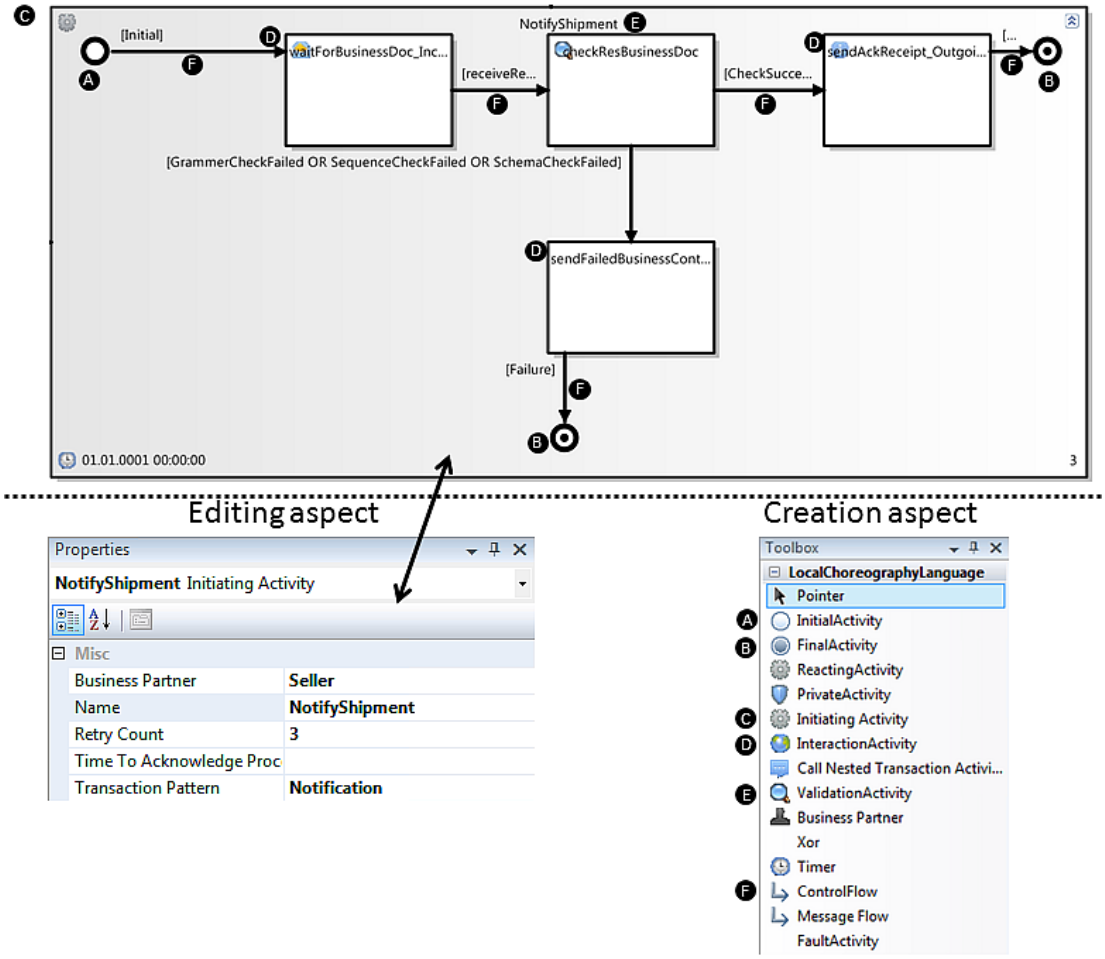

Abbildung 3.9: The local choreography language zoomed in on the transformation of the NotifyShipment business transaction from the example scenario for the buyer

Figure 3.9 anticipates parts of the resulting local choreography coming from the transformation of the "order from quote" example outlined in section 3.2.2. It illustrates, that the LCL is in many ways similar to the BCL but more concrete in certain defining aspects. Fundamentally LCL models are still technology independent descriptions of business logic. Similar to the BCL, the LCL is also a composition of single entities out of a finite set of different types showcased in the toolbox window screenshot in the lower right half of figure 3.9. Also each entity instance on the editing canvas can be selected to reveal type specific properties in the properties window of BSopt Designer. One aspect that's not been used in the BCL is the possibility of defining parent/child relationships. In the LCL, *business transactions* are generally represented as parent shapes nesting all child entities acting within their boundaries. This change of representation presents the opportunity to specifically outline the different actions to be taken by the participant while in course of a business transaction. The elements used inside a business transaction are realizations of concepts suggested in [19] which presents a strategy to translate UMM business transactions into state machine based graph representations for local choreographies. As the BCL itself is defined as a projection of UMM concepts into the area of domain specific languages, the resulting elements demonstrated in [19] were well suited as starting point for the LCL. One consequence of this design is that strictly

speaking some aspects of the LCL such as validation fall into the area of orchestrations. Despite this circumstance the communication with a business application where most of the internal logic is processed is abstracted in this process. Thus for this thesis we will keep referring to the LCL as a meta model for local choreographies.

The following enumeration addresses the types of entities used in the transformation process to create LCL models. The bracketed characters are referring to elements from figure 3.9:

- *business transaction* instances are defined by the *reacting-* and *initiating activity* (C) types. The only difference between those is the indication whether the participant is in a reacting or initiating role. This difference represents whether the first message exchange action turns out to be based on a receiving or sending activity. Additionally *business transactions* expose properties such as their original *business transaction pattern* and a maximum retry count. This property is adopted from the definition in [16] and tells how often a requesting authorized role must re-initiate the business transaction in the case of a timeout exception. The exception will be thrown when the time to acknowledge a receipt or processing or time to respond has been exceeded.

- the *initial activity* type (A) represents the starting point of the control flow both on a global level and inside each *business transaction* instance where it must occur exactly once.

- the *final activity* type (B) defines end points for a *business collaboration* on a global level and inside each *business transaction*. They are attributed to define either a business failure or success.

- the *interaction activity* type (D) is responsible for message exchange in different contexts. It possesses an *interaction type* property which may either be "ingoing" or "outgoing" thus defining whether it symbolizes a message receiving- or sending action. Additionally, its *activity type* property must have one out of four different values:

  1. the value of "acknowledgment" makes the activity send or receive an acknowledgment message.
  2. the value of "message" relates to the in- or outgoing handling of a business document.
  3. the value of "business application" refers to communication with the business application indifferent of set *interaction type*. This indicates that a business application must infer a business decision.
  4. the value of "error message" is used to communicate an impending business failure.

- a *control flow* element (F) is used to connect other entities on a global perspective or inside a *business transaction*. The entity possesses a *control flow outcome* property which may either be unspecified, indicate a success or a failure. The *validation activity* uses this property to branch based on successful or failed validation as do *business transactions* based on their business outcome.

- the *validation activity* (E) is another multi contextual entity. Its *validation type* property may be one of the following:

1. the *check retry count* value validates whether there is a retry left after re-initiating the business transaction. This is used so a participant will not try to endlessly contact a non reachable or technologically impaired business partner.
2. the *check ack receipt* and *check ack processing* values are used to scan the content of a specific acknowledgment message and branch accordingly if an error is being detected in the course of this process.
3. the *check res business doc* value symbolizes the data validation for a business document with checks such as grammar-, sequence- and schema validation [19].
4. the *validate content of res business doc* value specifies the validation of a business document on a higher level, e.g. amounts of money being in a meaningful value range.

With the entities used in the transformation process explained we can return to elaborate on the business transaction in figure 3.9. The upper half of the illustration shows the result of transforming the *notify shipment* business transaction from its BCL representation (figure 3.8)into its LCL representation for the buyer role. It consists of a *reacting activity* acting as a parent for seven interconnected child entities. The internal control flow starts with the obligatory *initial activity* and continues with the business document receiving *interaction activity* called "waitForBusinessDoc_Incoming". It next advances to the *validation activity* named "check res business doc" with identical validation type. After that it either branches to another interaction activity in order to acknowledge the receipt of the business document or ends with an *interaction activity* indicating an error after failing validation. The resulting business state will be determined by each concluding final *state activity*. The *notify shipment business transaction* has been chosen for figure 3.9 as it results in the least complicated and thus smallest business transaction but still explains the way an automatically generated LCL *business transaction* will look like. Figures 3.10 and 3.11 show the complete transformations of the BCL example for both, buyer and seller. The difference in size between the three *business transactions* is due to the different *business transaction patterns* used in the source choreography. As an example the *commercial transaction* pattern used for the legally binding "place order" business transaction includes the most acknowledgment handling and validation entities and, thus, is the most complicated transformation case.

The transformation process defined in [19] envisions the task of checking the retry count before sending a requesting business document. As this task is only performed in initiating local *business transactions* those feature a by one higher number of child entities as opposed to their corresponding reacting *business transactions*. Apart from this detail the transformation from a global choreography into local choreographies guarantees that the created structures are exactly complementary to each other. In other words this means that for each specific sending activity a compatible receiving activity has to be scheduled for the overall *business collaboration* to work flawlessly as expected.

3.2.4 Transforming a BCL model into LCL models

Having introduced the source and target representations of the example choreographies the transformation process connecting BCL and LCL models will be elaborated. This description is separated into two parts. First the general approach of transforming a

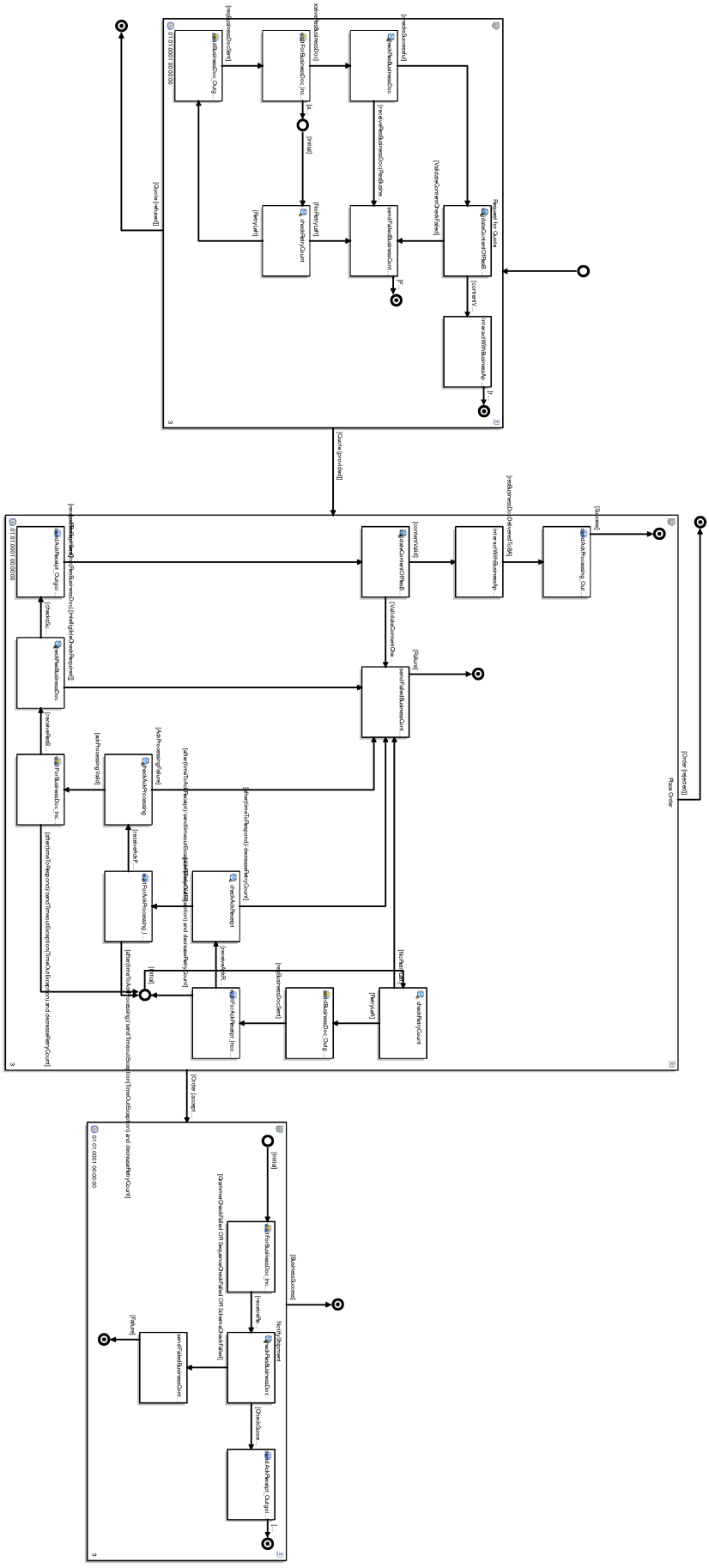

Abbildung 3.10: the “order from quote” example transformed to a local choreography for the buyer role

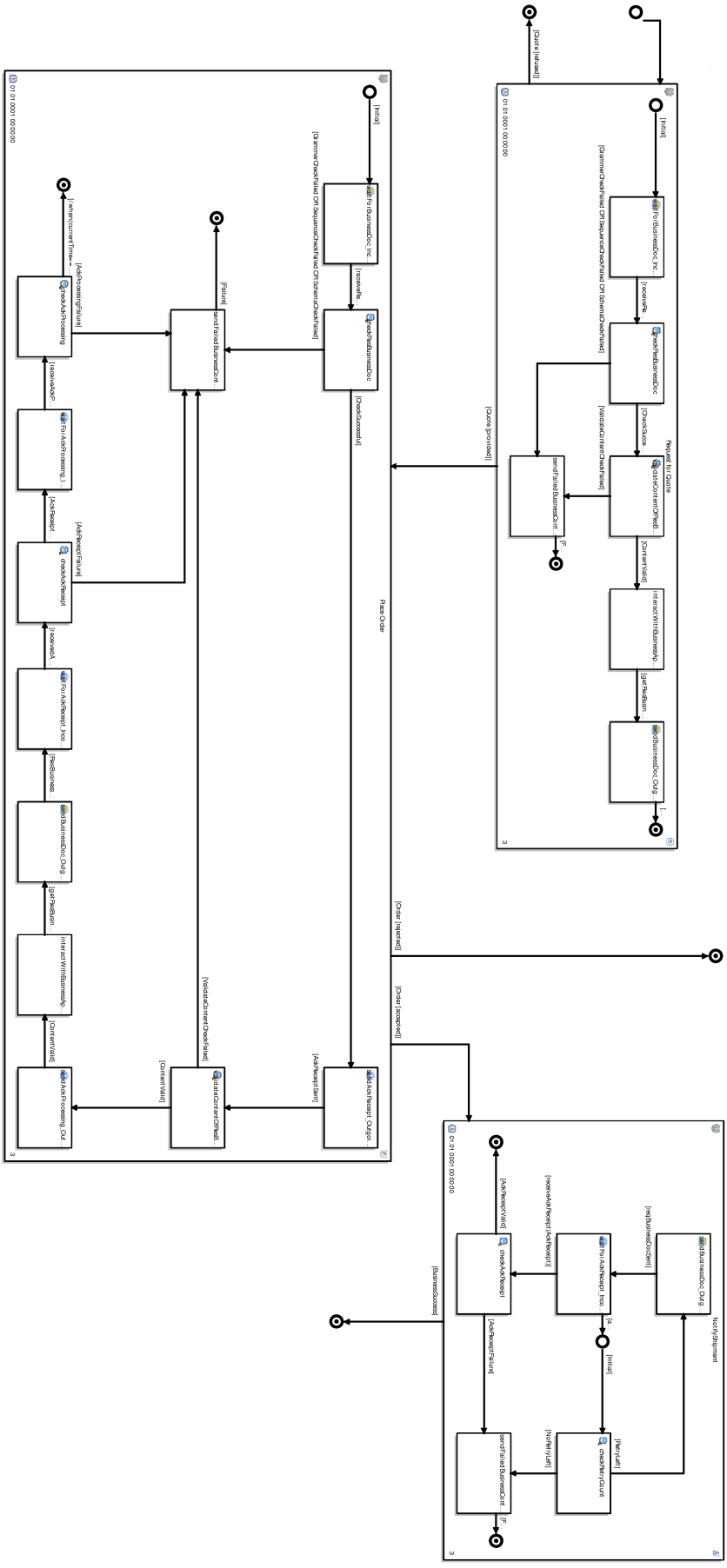

Abbildung 3.11: the “order from quote” example transformed to a local choreography for the seller role

Title: transform global choreography into local choreographies
Short Description: The user indicates the roles for which he wants a local choreography created and lets the system create those.
Preconditions: A solution must be opened and the source choreography must be part of a project in this solution.
Description of course of events: E1) The user launches the transformation wizard. A1) The system displays the transformation wizard showing an introductory information screen. E2) The user proceeds to the next step. A2) The system displays the 'Select Roles' step including all roles found for the specific global choreography as selectable entities. AA2) The system indicates why it is unable to process the input file and bails out, terminating the use case. E3) The user selects one or more roles and proceeds to the next step. A3) The system transforms the input file into local choreographies for each role selected and displays a summary of the process indicating the success or any errors or warnings traced in that process. E4) The user leaves the wizard A4) The system closes the wizard window
Consequences: A new local choreography file with file extension .lcl will be created for each selected role and added to the BusinessChoreographyView/LocalChoreographyView subfolder of the project the source BCL file is part of. If this subfolder does not exist it will be created. Additionally a .diagram file will be added for each generated .lcl file and added into the same target directory on the file system. Already existing .lcl and .diagram files will be overwritten without warning.
Notes: Users may choose to leave the wizard at any time before the transformation process begun without causing any consequences. Due to the repetitive nature these alternative events have been omitted from the description of course of events.

Abbildung 3.12: The "transform global choreography into local choreographies" use case.

global choreography within BSopt Designer will be highlighted. Afterwards, the following part will cover the specific way this transformation was implemented and integrated into the tool environment.

The steps a user has to consider in order to transform a global choreography into LCL models are best demonstrated in the form of a use-case description. Figure 3.12 demonstrates the necessary steps to accomplish this task. As demonstrated, the one relevant choice users face is the selection of the roles for which to generate local choreographies. The concrete implementation in BSopt Designer offers a context menu item for .bcv files acting as storage representation of BCL models in the tool's solution explorer. Once clicked the transformation wizard will be launched displaying the single steps visualized in figure 3.13 in the course of a normal sequence of events. The only dynamic content is shown in step 2 where the roles a user can select are taken from the entries of the expected single *participants* container from the input choreography.

The implementation of the transformation process extends the approach presented in [19]. Furthermore, it considers the different kinds of *business transaction patterns* described in [16]. As mentioned in section 2.5 three possibilities for model transformation were considered:

- Using the *T4 templates* technology in order to transform a given base template file into a new representation by accessing additional data from a source model.
- Working with the domain model API provided by the DSL tools infrastructure.
- Manual creation of the target DSL storage representation by creating and serializing a tree of suitable XML-constructs.

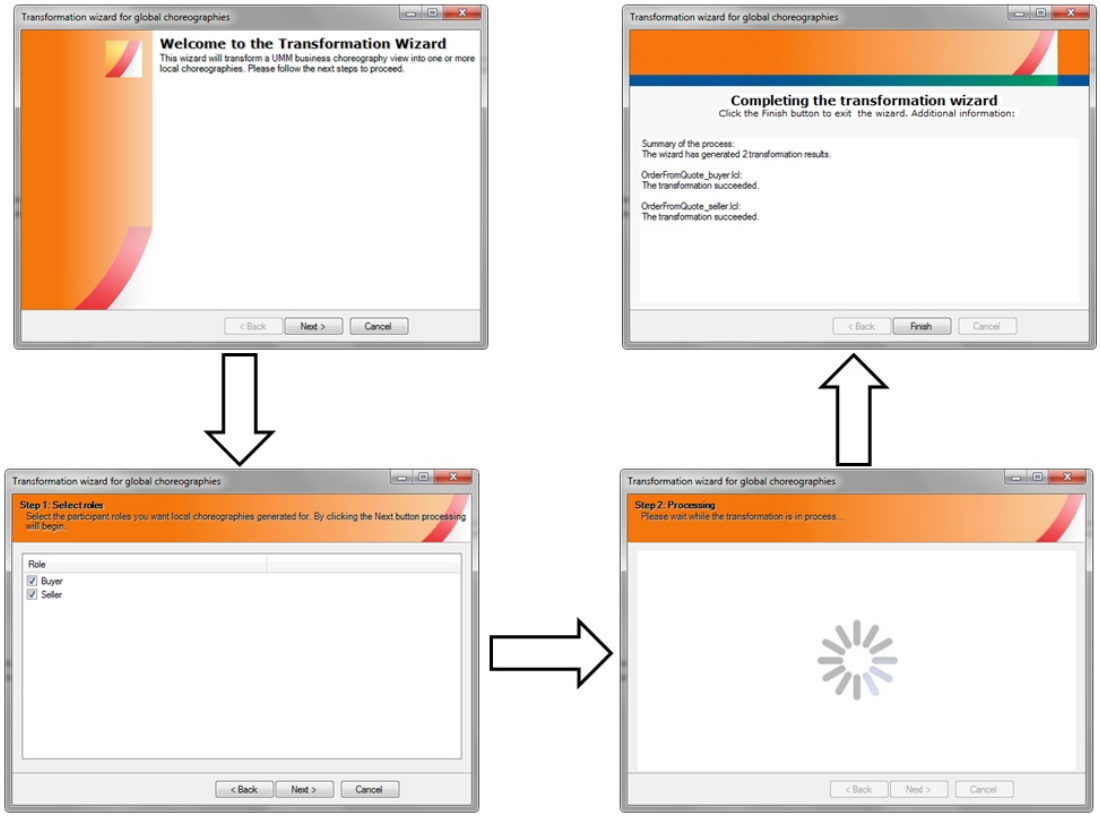

Abbildung 3.13: The transformation process for the “order from quote” sample

From these three approach (3) was dismissed quickly due to the unnecessary work and possible maintenance problems due to a still changing meta model. This is in contrast to the option of working with an always up-to-date object model provided by the DSL Tools system for free. While a transformation process using T4 principally would have been possible, there was little to gain with this approach as the embedding of information into a static template structure was at no point needed. Additionally, the tool support for creating text templates presented itself less mature than the powerful ways to handle C# source code in Visual Studio. Finally, the handling of a text template in BSopt Designer to transform a .bcv file was much less straightforward compared to using the object model provided for source and target DSLs. The basic approach is thus to load the global source choreography, iterate over all *business transaction* objects in code, create corresponding constructs using the object model of the local choreography DSL and reconnect entities creating a comparable topology in the target DSL. Describing the resulting code in depth would require to elaborate about the specifics of the DSL Tools domain model and also cover a lot of code required to account for each specific business transaction pattern, certain properties to set and so on. As this approach would add little convertible information for most readers, the basic structure of the code has been condensed into the function *TransformGlobalChoreography* defined in algorithm 1 which is expressed in pseudo-code. The resulting entities transformed from global *business transaction activities* are based on the state machine depictions presented in [19].

After a local choreography instance for a specific role has been devised, it must be serialized into its storage representation. The DSL Tools object model provides helper classes for both serialization and deserialization out of the box. Additionally as mentioned in the use case description in figure 3.12 each local choreography must be associated

with a .diagram file. This file separates the layout and sizing information saved for the visual representation of a DSL from its data part. While the layout for user created diagrams is generated based on user actions while editing the models it's less straightforward when creating a model from scratch in code. The core object model of DSL Tools offers limited auto layouting facilities which only support the layouting of global shapes. In other words while this facility sufficed to create well looking BCL models it was not adequate to build local choreographies including business transaction shapes nesting a number of child activities. In its default realization all nested *business transaction* child shapes in a local choreography were positioned at the upper left corner of the parent and had to be repositioned exhaustingly by users in order to visualize the devised process flow.

Manually layouting nested diagram elements in an LCL model

As the default layout produced for LCL models was generally considered unacceptable the logical conclusion was to create a custom fitting layout supporting the nesting of child shapes. This was possible as the DSL Tools object model provides the user with both the logical relationship between activities matching a directed graph representation and sizes of activity shapes which are a natural factor when considering the layout of a graph[1]. As the layout generation for graphs is an active research topic in its own right the logical conclusion was not to start inventing or duplicating known algorithms but to rely on the experience of others. The requirements for the LCL model were defined as follows:

- The component must be provided as a library or set of source files released free of cost under a (preferably open source) license not restricting BSopt Designer from being released under the *GNU General Public License, Version 3.0* [10] when used in the product.
- The visualization of LCL models is based on shapes of different sizes connected by angular arrows. A suiting component must support these visual traits in its layouting algorithm and support finding a constellation which minimizes intersecting control flow elements while keeping the resulting graph reasonably compact.
- Interoperation with the library shall be reasonably straight forward. The best case scenario would be a library supporting direct usage under .NET.

After looking at various graph layouting libraries provided on the world wide web it showed that a library optimally satisfying all three requirements was probably not existing right now. Many high quality libraries were commercial, based on the java programming language or simply did not support explicit orthogonal layouting approaches. The best fit for this scenario presented itself in the *Open Graph Drawing Framework (OGDF) [59]*. As described by its authors OGDF is *"a self-contained C++ class library for the automatic layout of diagrams. OGDF offers sophisticated algorithms and data structures to use within your own applications or scientific projects. The library is available under the GNU General Public License. "*. While a C++ library cannot be accessed directly from a .NET environment there are 3 basic ways to enable interoperation with unmanaged code [35]:

- *Platform Invoke ("pinvoke")* is a mechanism which allows to call unmanaged functions exported by a dynamic link library (DLL) by defining and attributing the

[1] Parent shape sizes are of course dynamic and depending on the bounds of their children.

signature of the function as seen in a .NET environment with platform specific information such as calling convention, name mangling settings and the concrete name of the function's entry point. Data transferred between the two environments is marshalled on the managed side according to a default behavior or explicit attribution including how exactly to marshal string values, where exactly fields in an unmanaged structure representation are positioned and many more details like that. While a relatively simple approach, *pinvoke* is best suited for integrating procedural code into a managed environment.

- *C++ interop* also called *It Just Works (IJW)* builds on the ability of the Visual C++ compiler to process both unmanaged and managed code. By enabling CLR compilation it's possible to create "mixed assemblies" including both unmanaged and managed code which offers a simpler way of interoperation as managed code may access unmanaged data and vice versa. The usual way to provide unmanaged code to a .NET environment by using *IJW* is to offer managed wrapper classes which internally reuse an unmanaged object model exported by the original unmanaged library.

- upport for the *Component Object Model (COM)* allows C# to directly consume COM components and also extend its services into the world of COM itself. Specialized tools such as the "type library importer" application support the automatic creation of interop assemblies which can be used from .NET just like any other managed code.

As *OGDF* is not built to expose COM components the approach taken was to statically link the library with a new CLR supporting C++ project creating a mixed assembly for consumption in C#. One reason for this decision was that *OGDF* is not offered as a DLL in the first place which means that adapting it for *pinvoke* would have taken more time and hardly provided any benefits over *IJW*. While *OGDF* itself supports many layouting algorithms and different ways to treat graphs such as the specialized handling of colored subgraphs, weighted nodes and edge styles the requirements for LCL layouts were relatively simple. The approach taken was to look at the "orthogonal layout" sample presented on the library's site, first duplicate reasonable results for a test choreography using a native C++ application and then wrap only those classes needed to repeat the results from C#. The resulting "ManagedOGDF" library was then used in the transformation process to create reasonable layouts as were presented in figures 3.10 and 3.11. The approach taken to yield results was defined by recursively looking at any possible children of a shape and map the according graph topology and node dimensions from the LCL object model into an *OGDF* representation. Next we let the library find a suitable new layout and retranslate the data back to dimensions used in the LCL model. Parent shape dimensions are adapted based on their resulting child layout bounds.

Algorithm 1 Transformation of a global choreography into a local choreography for the role specified

```
1: function TransformGlobalChoreography(globalChor, role)
2:     localChor ← new local choreography
3:     ➢ Transform business transactions:
4:     for each business transaction bt in globalChor do
5:         if bt includes participant with role then
6:             localBt ← TransformGlobalBusinessTransaction(bt, role)
7:             localChor.AddBusinessTransaction(localBt)
8:         end if
9:     end for
10:    ➢ Transform final+initial activities:
11:    for each final activity finalAct in globalChor do
12:        localChor.CreateCorrespondingFinalActivityFor(finalAct)
13:    end for
14:    localChor.CreateCorrepondingInitialActivityFor(globalChor.InitialActivity)
15:    ➢ Rewire global activities:
16:    for each activity localAct in localChor do
17:        globalAct ←source activity from globalChor for localAct
18:        rewire localAct according to globalAct's direct connections
19:    end for
20:    return localChor
21: end function
22:
23: function TransformGlobalBusinessTransaction(globalBt, role)
24:    if role is initiating in globalBt then
25:        return TransformGlobalBtForInitiant(globalBt)
26:    else
27:        return TransformGlobalBtForResponder(globalBt)
28:    end if
29: end function
30:
31: function TransformGlobalBtForInitiant(globalBt)
32:    localBt ← new local business transaction
33:    ➢ Based on the business transaction pattern of globalBt
34:    ➢ localBt will receive different child activities here.
35:    ➢ basic structure: first send business doc, then optionally receive reply
36:    return localBt
37: end function
38:
39: function TransformGlobalBtForResponder(globalBt)
40:    localBt ← new local business transaction
41:    ➢ Based on the business transaction pattern of globalBt
42:    ➢ localBt will receive different child activities here.
43:    ➢ basic structure: first receive business doc, then optionally send reply
44:    return localBt
45: end function
```

3.3 Processing of message type descriptions

The electronic message exchange between partners involved in a business process is relying on an explicit understanding of the specific kinds of messages expected and how these messages are structured. In order to enable conformance in this area, message types have to be predefined so they can then be adopted in a process definition. As presented, global choreographies modeled using the BSopt approach define any message exchange by linking business documents to their *business transaction* activities and separating between requesting and responding information. The business document definition for BSopt Designer relies on the *Core Components* [60] standard which finds its integration into the tool by a collection of DSLs as described in [23]. The result of a business document modeling process is a set of XML schema files defining one business document made up of simple typed properties and/or more complex container types.

Business documents need to be integrated into workflow artifacts automatically. This problem definition means that in its most minimal form a workflow must incorporate given XML schemas for referenced business documents so it can validate the structure of incoming data. Additionally, a way to integrate schematically correct outgoing messages into the process has to be devised. For the concrete target workflow system "Windows Workflow Foundation 4.0" the strategy to solve this task has been aligned with the possibilities of the framework: WF4 relies on WCF services in order to send data over the wire. Messaging information involved in this process is always based on a "data contract" which is *"a formal agreement between a service and a client that abstractly describes the data to be exchanged"* [32]. Serializable .NET types, even when not attributed explicitly to act as data contract can be used in WCF to define a mutual understanding of the data structures exchanged between client and server. The sending of real object representations instead of one string containing XML comes with the advantage of automatic grammar and schema validation. Thus, not well-formed and invalid documents are rejected by WCF. Furthermore, the data access inside the workflow is greatly simplified as sent or received messages ultimately are stored in variables and dealt with in the same way as any other object in the system. For this setup to work the given XML schemas have to be turned into serializable .NET types. Those types can then be used in the workflow artifact creation process to act as technical realizations of the original business documents.

3.3.1 The business document transformation wizard

Before describing the steps which lead to the realization of a transformation process which is able to transform business document schemas into managed types, it's illustrative to look at the final result. Figure 3.14 illustrates the use case scenario which leads the user to the generation of a message type assembly. The wizard usage resembles the wizard for transforming global choreographies. There's only one dynamic step in which the user has to select the business documents he wants included in the final result, a processing phase and a summary screen. Figure 3.15 visualizes this sequence for the "order from quote" sample presented in section 3.2.2. The resulting "MessageTypes.dll" assembly will be used in the final wizard for creating a workflow, which is discussed later.

3.3.2 Evaluation of XSD transformation tools

Having outlined the motivation for creating a .NET type assembly out of XML schema descriptions, we now concentrate on the actual implementation. While a feasible

Title: transform XML schemata into a .NET message type assembly
Short Description: The user indicates the business document describing XML schemata he wants transformed into a message type assembly and lets the system create it.
Preconditions: A solution must be opened and include a process flow perspective project which includes one or more derived XML schemas in subfolders of the BusinessInformationView/DocLibrary folder.
Description of course of events: E1) The user launches the transformation wizard. A1) The system displays the transformation wizard showing an introductory information screen. AA1) The system informs the user that no XML schema files have been detected and the wizard terminates. E2) The user proceeds to the next step. A2) The system displays the 'Input Selection' step including all business documents which can be added into a .NET assembly E3) The user selects one or more business documents and proceeds to the next step. A3) The system transforms the selected corresponding XML schema files into a .NET assembly representing each business document as type in a dedicated namespace. The system displays a summary of the process indicating the success or any errors or warnings traced in that process. E4) The user leaves the wizard A4) The system closes the wizard window
Consequences: A new local message type assembly will be created or overwritten in the project's BusinessInformationView folder.
Notes: Users may choose to leave the wizard at any time before the transformation process begun without causing any consequences. Due to the repetitive nature these alternative events have been omitted from the description of course of events.

Abbildung 3.14: The "transform XML schemata into a .NET message assembly" use case.

approach would be to manually create code representations out of XML based type descriptions, general consensus was that first this approach might consume a lot of implementation time until yielding usable results and secondly was pretty complex and thus had to be tested rigorously. As free tools for deriving code from XML schemas are available the sensible approach was to evaluate those first before devising any code. The three tools tested were the XML schema definition tool xsd.exe [29], the serviceModel metadata utility tool svcutil.exe [31] and the LINQ to XSD [40] library.

The XML schema definition tool xsd.exe is a standard tool coming with the .NET framework since version one and can be used to generate *"XML schema or common language runtime classes from XDR, XML, and XSD files, or from classes in a runtime assembly"*. While generated classes are defined as *serializable*, *partial* so they can be extended externally and often of good quality there are XML schemas which lead to poor results or cannot be processed such as the XSDs offered as part of the BPMN 2.0 Beta 2 specification [51].

The serviceModel metadata utility tool svcutil.exe has been introduced together with WCF and is most commonly linked to automatic class generation based on WSDL input sources. What's less known is that it can also derive code from XML schemas. The testing of sample schemas showed that the utility is generally able to process our business documents and create compilable C# source code out of it. But our tests also revealed that the tool has its limits and tends to give up on complex schemas, creating C# classes with a very generic *Nodes* property of type *System.Xml.XmlNode[]* to act as generic container for any data. The efficient handling of generated classes definitely requires strongly typed structures which was the reason for abandoning this tool.

LINQ to XSD is an open source community project originally started at Microsoft aiming to build upon the existing LINQ to XML technology introduced with the .NET

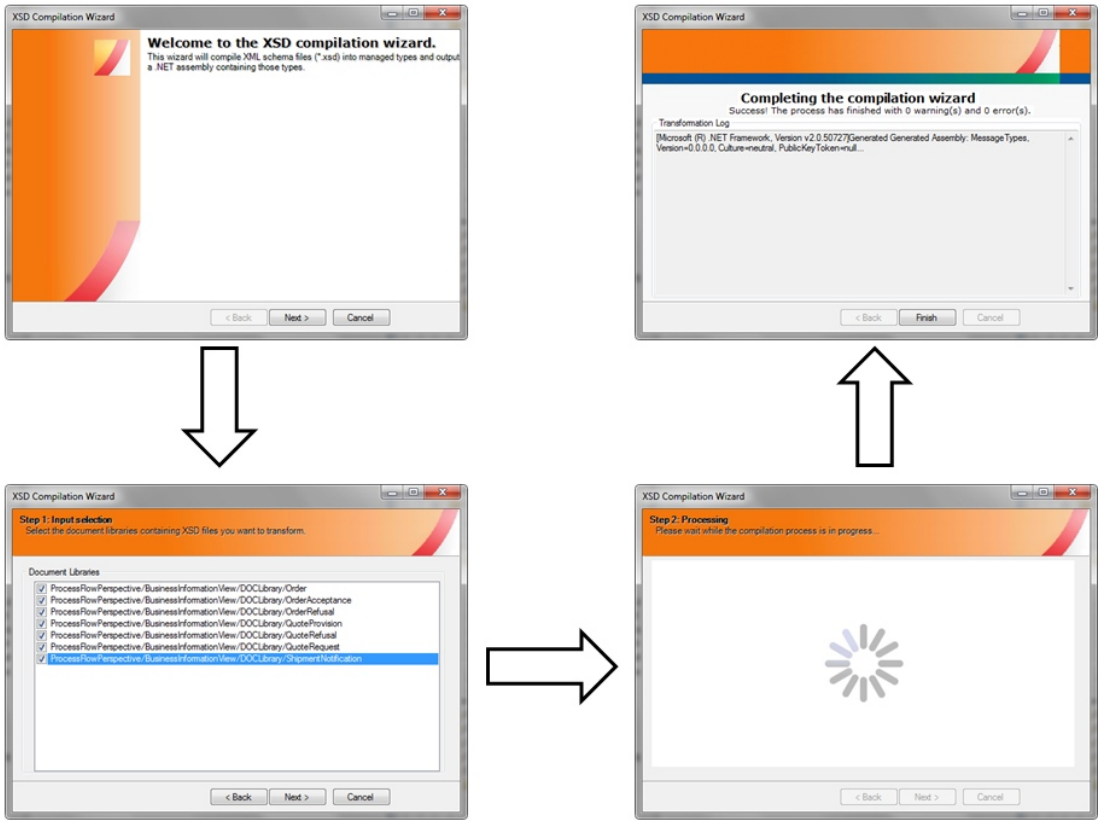

Abbildung 3.15: The XSD transformation process for the “order from quote” sample.

Framework 3.5. It consists of a library offering the basic framework and comes with a console application which creates typed wrapper classes and readily compiled assemblies from XML schemas. The output generated was generally showing the highest quality, offered convenience methods such as static load and save functionality and even included documentation into the source code when applicable. Figure 3.17 shows the most important elements created from the *shipment notification* business document as a class diagram coming from the XML schema definition portrayed in figure 3.16. It shows that the strategy used by LINQ to XML is not to create completely independent types but base their functionality on a common framework. This design avoids code bloat but introduces a new dependency which was deemed acceptable for BSopt.

As the output by the LINQ to XSD console application generated usable CLR based assemblies the design decision was to include the project with BSopt Designer and use the console application in the transformation wizard introduced earlier in order to generate the message type assembly. This also resulted in a pretty straightforward processing step which consists of the building of appropriate command line arguments and the execution of the LINQ to XSD utility with this information.

3.3.3 Business document instance creation

As we have seen the adoption of business documents into WF4 workflows can be solved by transforming any document structure into a .NET type representation ready to be plugged into a workflow definition. While this approach covers the validation and receiving of messages, the sending of data is inherently linked to document instances

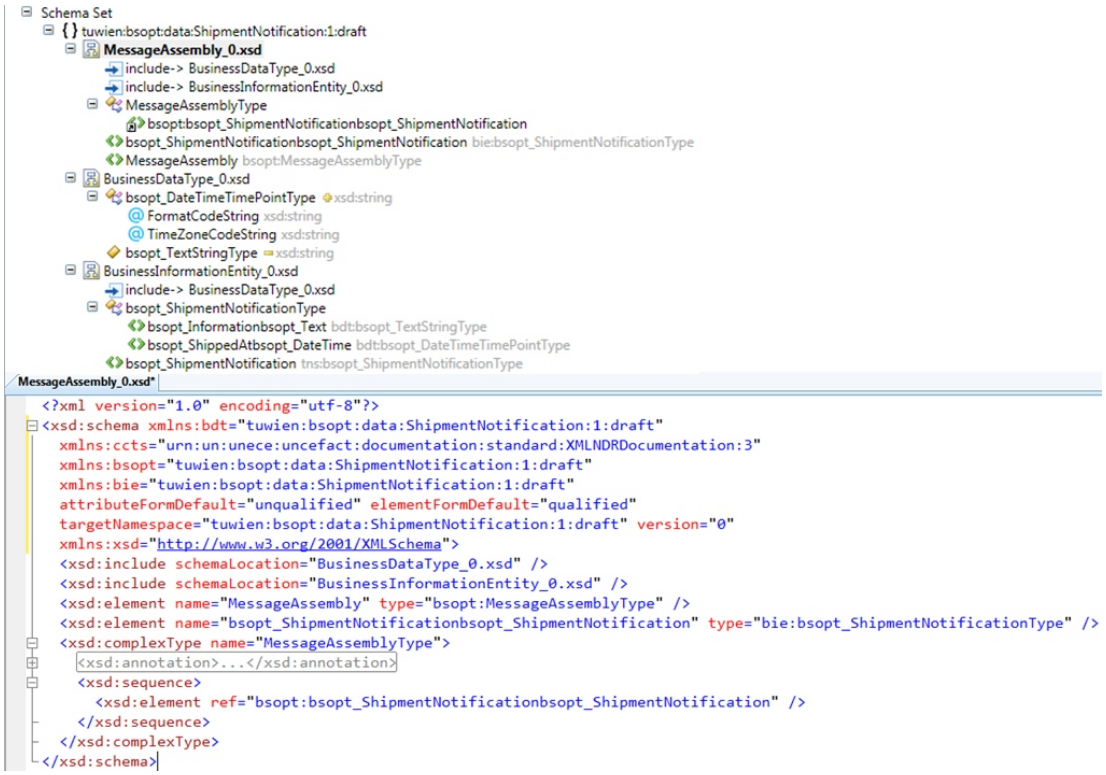

Abbildung 3.16: Schematic view of three generated XSD files describing the ShipmentNotification business document for the "order from quote" example.

conforming to the same business document structure. Workflow hosting business applications or users need to generate suitable XML documents prior to the workflow sending them. Listing 3.1 shows an example XML instance conforming to the schema defined for the *shipment notification* business document from the *order from quote* example scenario.

```
<?xml version="1.0" encoding="utf-8"?>
<MessageAssembly xmlns="tuwien:bsopt:data:ShipmentNotification:1:draft">
  <bsopt_ShipmentNotificationbsopt_ShipmentNotification>
    <bsopt_Informationbsopt_Text>The order has been shipped.</
        bsopt_Informationbsopt_Text>
    <bsopt_ShippedAtbsopt_DateTime FormatCodeString="dd.MM.yyy␣HH:mm:ss␣"
        TimeZoneCodeString="W.␣Europe␣Standard␣Time">31.07.2010 10:50:45</
        bsopt_ShippedAtbsopt_DateTime>
  </bsopt_ShipmentNotificationbsopt_ShipmentNotification>
</MessageAssembly>
```

Listing 3.1: A shipment notification instance for the “order from quote” example based on a Core Components definition derived XML schema

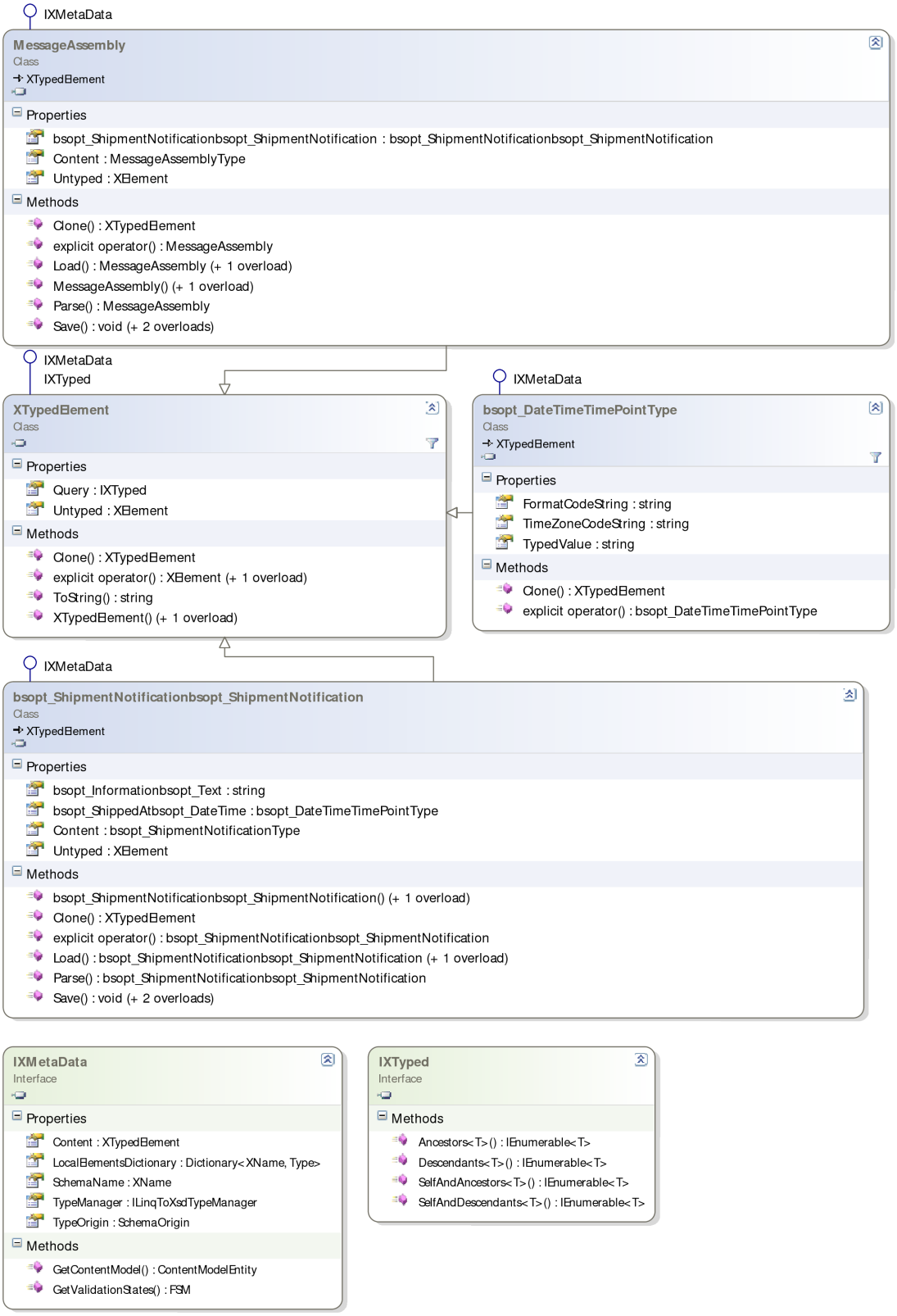

Abbildung 3.17: Created .NET class representation of the shipment notification business document

3.4 Generation of workflow artifacts

All steps previously described came with the motivation to serve as necessary intermediate stages in the generation of workflow artifacts. The purpose of this section is to describe this last element in the processing chain which builds upon a given local choreography, a fitting message type assembly describing all business documents to be used in the workflow and XML files defining specific messages to be sent. The choice to abandon WF 3.x and build upon WF4 which at the time of evaluation still was in a volatile beta 1 phase was motivated by the following facts as was also discussed in section 2.3.2:

- While WF3 supports the notion of message correlation which is necessary to route an incoming message to the right workflow instance, its realization "context-based correlation" is related to specific kinds of *contextful bindings* and practically the inclusion of a "context token" into any conversation initiating operation contract [44]. Contrary to this WF4 also supports "content-based correlation" where the problem can be tackled on a higher level without depending on specific data included as message headers.

- The control flow semantics WF 3.x supports are based on either a sequential or event based processing of activities. While event based workflows allow the representation of more complex business processes they are also harder to follow by humans compared to sequential workflows. While papers such as [61] show the possibility of transformations from graph based process definitions into sequential ones, they also quite evidently show that these strategies are more complex compared to any direct mapping approach. As WF4 comes with the new graph based flowchart control flow, it can be used in a more straightforward mapping approach to represent given local choreographies. This specific control flow pattern supports the transformation of any given activity in a local choreography into workflow specific constructs while the control flow semantics on the workflow side stay the same.

- The data handling perspective in WF3.x is based on fields and properties defined in code-behind files with data flow being realized by binding to these in-code definitions using dependency properties. While this approach may still allow a declarative workflow definition strategy when using specific custom activities its realization is challenging. WF4 introduces a completely declarative approach by incorporating the definition of variables and the accompanied data flow into the workflow specification. This design supports the generation of self contained workflow artifacts and lets us avoids the problem field of generating custom code on the fly.

- As with data handling, WF4 also supports the declarative definition of execution logic by introducing its own *Visual Basic* based expression language. Specific logic can thus be included into a generated workflow definition itself where a code behind file was necessary in WF3.x[2].

- In WF3 it was necessary to define service contract interfaces which define the desired data exchange semantics to be performed by the workflow. Defining this

[2]It shall be noted that WF3.x comes with its own rules engine which can replace in-code logic at some places. Nevertheless the applicability of this technology is much more limited compared to WF4 expressions.

information for dynamic operations and creating binaries out of these definitions would have introduced a new area of complexity into the system including all the disadvantages that come with it. WF4 replaces this approach by an automatic service interface deduction mechanism. This approach was much more desirable as it meant that defined workflows just work based on the declarations created by the workflow artifact generator.

With these preliminaries out of the way this section is structured by first introducing the usage perspective of the workflow generation wizard. Secondly, an elaboration of the actual transformation implementation is presented and thirdly, certain specifics considered noteworthy are discussed.

With workflow artifact generation the gap between technologically independent process description and concrete implementation is closed. This transgression toward a specific technology comes with concrete requirements for additional information to be entered by the end user. In case of Windows Workflow Foundation 4, these missing input tokens needed to successfully transform a local choreography model are threefold:

1. One required data item is the specific WCF binding used for communication with business partners. This also defines the mode of communication which as example includes message queues, message transportation over HTTP or binary transportation.

2. The specific endpoint information for all business partners has to be given so it is known where to contact process participants.

3. All message based interaction activities from the input choreography have to be correlated with specific business document types. Sending interaction activities also need to be linked with specific message instances.

Figure 3.18 outlines the corresponding use case which describes a user inputting those informational fragments into a wizard in order to generate workflow artifacts. As an accompanying illustration, figure 3.19 shows the actual user experience of the workflow generation wizard. As shown, users first select the choreography to transform and which message type assembly deemed suitable for the actual message exchange. Next they can decide whether to input workflow specific data by using the wizard or by using a predefined XML file holding equivalent information. Assuming users choose to input data manually they now assign each incoming activity from the source choreography one or more business documents taken out of the message type assembly provided. As an example the buyer from the "order from quote" scenario might receive a *quote provision* or a *quote refusal* message when requesting a quote in the "request for quote" *business transaction*. The following step in the wizard requests similar information for outgoing activities defined in the source choreography. This time users additionally need to provide the actual content of outgoing business documents which are to be provided in the form of XML files as described in section 3.3.3. Again one or more business documents can be associated with one interaction activity. At last users have to specify the concrete WCF binding information governing *"transport, encoding, and protocol details required for clients and services to communicate with each other"* [32]. Additionally a "base address" has to be specified for the workflow services for all process participants. The wizard automatically derives fitting endpoint addresses for all messaging activities specified in the source choreography thus alleviating users of having to specify each and every endpoint address and linked service contracts manually as would be necessary when defining the workflow in a traditional way.

Title: transform a local choreography with message type assembly and XML messages into workflow artifacts
Short Description: The user indicates which local choreography he wants transformed into a workflow and how and lets the system create workflow artifacts ready for hosting in a suitable business application.
Preconditions: A solution must be opened and include an execution perspective and process flow perspective project. The latter must include at least one .lcl file and a message type assembly.
Description of course of events: E1) The user launches the transformation wizard. A1) The system displays the transformation wizard showing an introductory information screen. E2) The user proceeds to the next step. A2) The system displays the 'Basic Settings' step offering sources for local choreographies and message type assemblies. AA2) The system informs the user that no local choreography or message type assembly have been detected and the wizard terminates. E3) The user selects a local choreography and a message type assembly and proceeds to the next step. A3) The system shows the 'UserInput.xml file' step. E4) The user selects whether he wants to create a new userinput.xml file or chooses one in the file system compatible with the current process. A4) In case the user chose to create a new userinput.xml file the system displays the 'Ingoing activities' step, else the use case skips three wizard steps and continues with (A7). E5) The user links each ingoing message activity from the local choreography selection with a type from the message type assembly and proceeds to the next step. A5) The system shows the 'Outgoing activities' step. E6) The user enters a directory on the file system harboring all XML documents prepared for sending by the generated workflow and links each outgoing activity from the local choreography selection with a type from the message type assembly and a corresponding XML document for the path entered before and proceeds to the next step. A6) The system shows the 'Binding and base addresses' step. E7) The user selects a binding to be used for the workflow, enters a base address for all services provided by the workflow, enters base addresses for all participants given by the local choreography and proceeds to the next step. A7) The system shows the 'Finished collecting data' step. E8) The user acknowledges that he's about to create a new workflow and overwrite any old data and proceeds to the next step. A8) The system shows the 'Transforming' step and after processing is done automatically shows the 'Completion' step. The system displays a summary of the process indicating the success or any errors or warnings traced in that process. E9) The user leaves the wizard. A9) The system closes the wizard window.
Consequences: The execution perspective receives a new subfolder named after the selected local choreography which includes a generated .xaml workflow definition, an application configuration file, all XML documents and supporting assemblies and other files.
Notes: Users may choose to leave the wizard at any time before the transformation process begun without causing any consequences. Due to the repetitive nature these alternative events have been omitted from the description of course of events.

Abbildung 3.18: The "Transform a local choreography with message type assembly and XML messages into workflow artifacts" use case.

Abbildung 3.19: The workflow artifact generation for the buyer in the "order from quote" example

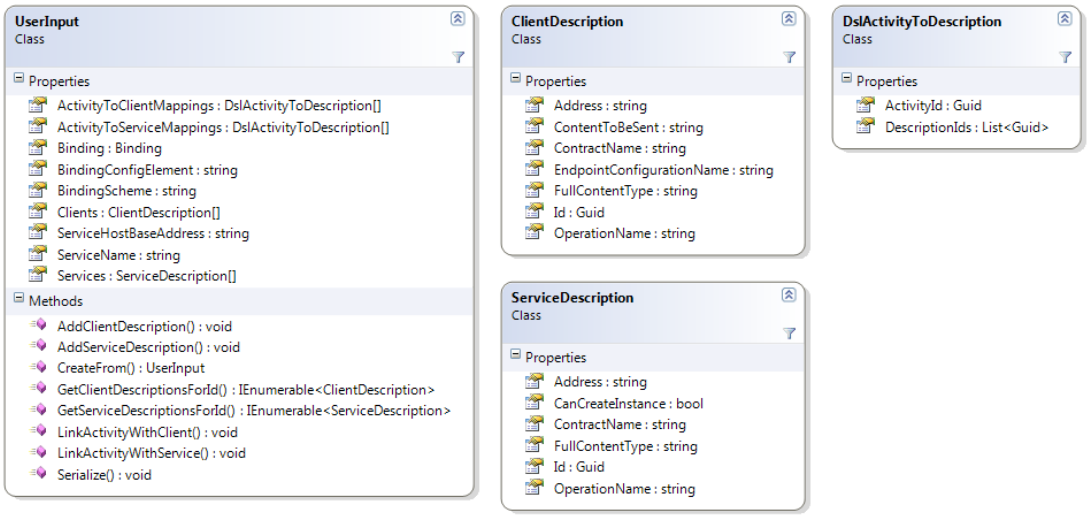

Abbildung 3.20: A class diagram representing types used for userinput.xml serialization

The concrete outputs the wizard generates are a workflow definition based on the *XAML Object Mapping Specification 2009* [42], an application configuration file defining necessary endpoint specifics to be applied by hosting business applications and files supporting the automatic hosting process. Those consist of assemblies needed for type resolution (*LINQ to XSD* base types and message types) and copies of given XML documents.

3.4.1 Implementation of the workflow artifact generation process

The WF4 technology is based on the *Common Language Runtime* version 4 acting as Microsoft's implementation of the Common Language Infrastructure (CLI) standard [4]. While this new version comes with many improvements and provides new feature possibilities to CLR based languages it also means that programs written for older versions of the CLR cannot directly interoperate with CLR 4 based data structures. BSopt Designer, building upon the Visual Studio 2008 Shell is developed using C# 3.0 which uses libraries provided with the .NET Framework 3.5 and is supported by the CLR version 2.0. This means that it's not possible to directly work with the data structures necessary to serialize WF4 artifacts from within the BSopt Designer process. Fortunately there is backward compatibility from CLR 4 to CLR 2 which means that e.g. class libraries written in C# 3.0 can be loaded into a CLR 4 based process [21]. Exploiting this possibility the general concept for the workflow artifact generation process is based on building a CLR 4 based console application taking input data such as described in the previous section and using this information to create the desired output data.

The resulting *local choreography transformer* application *LCT.exe* takes just one command line argument which defines the transformation directory on the file system. However this directory must include exactly one .LCL model file, a message type assembly, a userinput.xml file, additional assemblies needed for type resolution and all needed business document representing XML data files. When using the workflow transformation wizard within BSopt Designer, those files are automatically copied into a destination folder before calling the actual transformer. The rest of this section is separated into

three parts. First the contents saved in a userinput.xml file are given a closer look. Secondly, the actual transformation steps are explained. Thirdly, the section is finished by an observation of the creation process of the co-created application configuration file.

The data stored inside any userinput.xml file is basically a serialization of the classes involved in figure 3.20, which come from a class library shared between BSopt Designer and *LCT.exe*. The *Userinput* class stores the information users have to enter when using the workflow transformation wizard in BSopt Designer: apart from single properties concerning the used binding, and base addresses its main responsibility is to link messaging activities defined in the local choreography to either service- or client descriptions. Which of these description classes are to be used is depending on whether the original activity is defined as in- or outgoing. Each activity in the source choreography is identified by a *globally unique identifier* (GUID) as is each client/service description and those ids are used in the linking process as modeled in the *DslActivityToDescription* class. Apart from the data a user is entering explicitly, additional information is automatically derived in order not to overwhelm users with technical details while only sacrificing minimal technical flexibility:

- An *operation name* is stored inside the description classes as part of the implicitly set up *WCF service contract* which is automatically defined to be either "SendMessage" or "SendAcknowledgement".

- Endpoint addresses are unique per *business transaction* and generated by adding the *business transaction* name, stripped of whitespaces and encoded to an XML local name, to the base address given by the user.

- A unique *XML local name* encoded *service contract name* is automatically derived from the current *business transaction* name and the specific kind of message. The algorithm for creating a service contract distinguishes between acknowledging- and business document handling activity types.

  It adds the character 'I' to the *XML local name* encoded name of the current *business transaction*. Then it concatenates the index of the messaging activity inside a *business transaction* based on its activity type and a character encoding the activity type itself ('A' for acknowledgments, 'M' for business document messages). Finally, it adds another character to distinguish alternative messages for the same activity in the source choreography. A workflow-wide unique example service contract would read "IRequest_x0020_for_x0020_Quote1Mb" and represent the 2nd business document messaging activity[3] in the "Request for Quote" *business transaction* acting as 2nd alternative.

- The *CanCreateInstance* property is needed when a workflow is defined to act as a "service workflow". Contrary to a "non-service workflow" which is to be launched manually a new service-workflow instance is only created by the workflow definition host when receiving a special first message. In the "order from quote" example the seller would host service-workflows as he's waiting for "QuoteRequest" messages while the buyer is launching non-service workflows. Receive activities in WF4 can launch new workflow instances but for this to work their *CanCreateInstance* property has to be set to `true` and the Receive activity must be the first activity inside a workflow definition. This is the reason for the *CanCreateInstance* property inside the *ServiceDescription* class which will only be set to true for the

[3] Indexing starts with zero.

first receiving activity inside the first *business transaction* of a service-workflow describing choreography.[4]

The actual transformation of a choreography into a workflow definition in *LCT.exe* can be separated into four steps which will described precisely in the sections following the enumeration:

1. Creation of an object representation from the local choreography storage model (*choreography loading*).
2. Transformation of *business transactions* and relinking of target workflow activities based on the source choreography graph topology.
3. Postprocessing of the created workflow definition on the object level.
4. Serialization of the workflow definition into its XAML based storage representation.

Transformation step one: choreography loading

While the handling of local choreographies in section 3.2.4 was based on the object model supported by *DSL Tools 2008*, the object model representation in the local choreography transformer application is not relying on it. Instead the class hierarchy has been rewritten to work by directly interpreting the .LCL XML storage representation, which is greatly facilitated by the *LINQ to XML* [39] base technology included with the .NET framework 4.0. The reason for this move is that Visual Studio 2010 does not directly support *DSL Tools 2008* but comes with an updated yet incompatible new version. It might have been possible to directly reference the local choreography's API generated with Visual Studio 2008. However, this approach was not further tested as the reality of switching development environments each time the DSL meta model was changed externally was considered too cumbersome and error prone to be practical. The version control system used by the BSopt project does not include generated binaries. Thus, each team member using the local choreography transformer would have to make sure individually, that the newest meta model assembly for the local choreography was referenced in Visual Studio 2010.

Transformation step two: activity transformations

Step two, the actual transformation process, is based on algorithm 2. The algorithm shows how a directed graph can be used as a source for creating a new destination graph representation with transformed nodes but the same topology such as the source graph. Lines 3 to 6 transform each source vertex from the source graph into a new representation (by calling ConvertToDstVertex()) which gets added to the destination graph. In a second pass lines 7 to 13 show the algorithm iterating through pairs of vertices connected by an edge in the source graph and applying an equivalent edge on the destination graph so to reconstruct the topology from the source graph representation.

As WF4 supports a flowchart based control flow it's possible to use this algorithm to transform the graph structure of a local choreography into a corresponding workflow definition. The actual implementation does this on two levels: first, the graph containing global elements representing either initial-, final- or transaction activities is converted

[4] In this context 'first' is based on the order of execution a choreography definition predefines.

Algorithm 2 General transformation of a source graph into a destination graph representation

```
1: function TransformGraphRepresentation(sourceGraph)
2:     dstGraph ← new destination graph
3:     for each vertex vert in sourceGraph do
4:         dstVert ← ConvertToDstVertex(vert)
5:         dstGraph.AddVertex(dstVert)
6:     end for
7:     for each vertex vert in sourceGraph do
8:         dstVert ← GetDestinationVertexFor(vert)
9:         for each vertex srcTargetVert in GetTargetsOf(vert) do
10:            dstTargetVert ← GetDestinationVertexFor(srcTargetVert)
11:            dstGraph.AddEdge(dstVert, dstTargetVert)
12:        end for
13:    end for
14:    return dstGraph
15: end function
```

and rewired to represent the same topology as the source graph. Initial- and Final- activities are converted into *WriteLine* activities to state the current state of the workflow. Transaction activities are transformed into *FlowChart* activities. Secondly, the child activities contained inside each source transaction activity are transformed again in the same fashion as on the global level. As *business transaction* activities in LCL hold objects all deriving from the common *ControlFlowElement* base class, a simple approach to transformation would be to introduce specific transformation operations based on the concrete type of an object using an ever extending if-else pattern. While this approach keeps the common concern of transformation on a local level, it clearly is problematic as changing object hierarchies are hard to maintain this way, especially when a code base starts to fill with this design at many different places. The approach of introducing a virtual method on the base of the type hierarchy in order to perform the transformation, while generally advantageous for solving type dependent concerns, clearly violates the *single responsibility principle* which states that *"a class or module should have one, and only one, reason to change"* [25]. However, transformations into workflow elements clearly are not the reason the classes have been defined for in the first place. One design pattern created to aid in scenarios where it's desired to non-intrusively extend a type hierarchy with external algorithms is the *visitor pattern* [11]. By introducing double dispatching, it enables calls which are depending on the runtime types of the processed objects in the type hierarchy and a visitor's concrete type. The pattern realization in *LCT.exe* is shown in figure 3.21, which shows the *Transformer* class associated with the *IControlFlowElementVisitor* interface and *ControlFlowElement* types. Each type deriving from *ControlFlowElement* has to implement the *Accept* method as shown in the Note in the illustration. A *Transformer* instance can then iterate over a collection of *ControlFlowElement* instances and call *Accept()* on each instance while handing over an *IControlFlowElementVisitor* implementing instance such as a *ControlFlowElementToActivityTransformationVisitor*. This results in two polymorphic dispatches with the first being the virtual call on the *ControlFlowElement*. The second resolves the particular method to be called on the visitor which works as the concrete type of the *ControlElement* is known from within the instance's *Accept* method which enables the

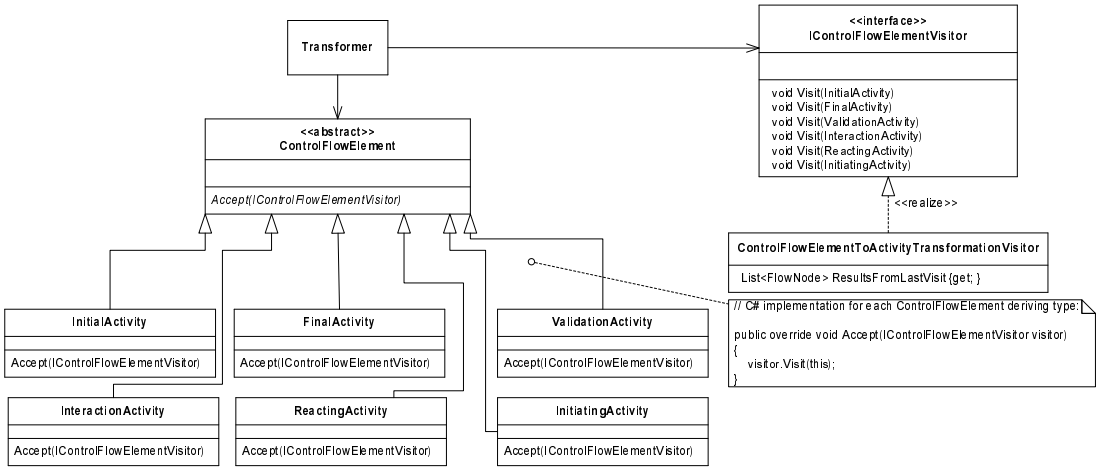

Abbildung 3.21: The visitor design pattern as realized in the *local choreography transformer* application.

right overload on the visitor to be called.

The *ControlFlowElementToActivityTransformationVisitor* offers the public property `List<FlowNode> ResultsFromLastVisit` with *FlowNode* describing an abstract base class for all nodes which can be included inside a WF4 flowchart based workflow. This property acts as destination vertex as defined on line 4 of algorithm 2 with the *ConvertToDstVertex* function realized by the usage of the just presented visitor implementation.

Before discussing the transformation of each child activity of a *business transaction* container within a local choreography, the structure of a resulting workflow definition must be defined. Figure 3.22 shows the result of the workflow transformation for the buyer in the "order from quote" example scenario. The layout has been done manually as an automatic layouting feature was out of scope. The basic structure of the workflow definition is very much comparable to the source choreography: the execution logic leads workflows into *business transactions*, which are represented as nested flowchart activities. It then makes them branch afterwards based on a switch construct which might very well encode more than just two cases as seen in the illustration. What's not directly displayed is that each switch statement on the global level is based on the evaluation of a global workflow variable of type *Int32* called *lastTransactionOutcome*. This variable is set inside each nested *business transaction* representing flowchart activity based on the individual business result. Table 3.1 lists this and all other global variables which are defined for a generated workflow.

The build up of flowcharts which are describing *business transactions* is dependent on the topology of the source choreography. This topology itself is the result of the selected *business transaction* pattern in the original global choreography and the specific role of the executing party for this *business transaction*. Despite this inherent volatility there are some variables which are defined on a per-flowchart basis. Table 3.2 outlines the name, type and reason for the definition of these.

As argued in the beginning of this section, one great advantage of the new version 4 of Workflow Foundation is that its design allows the straightforward definition of workflows in a purely declarative way. While the base activity library coming with the product helps tremendously in this realization, there are always cases when this is not enough. Workflows generated with BSopt Designer need to communicate with a hosting business application to enable human interaction for decision making. One strategy to implement this requirement is to rely on available WCF based communication activities

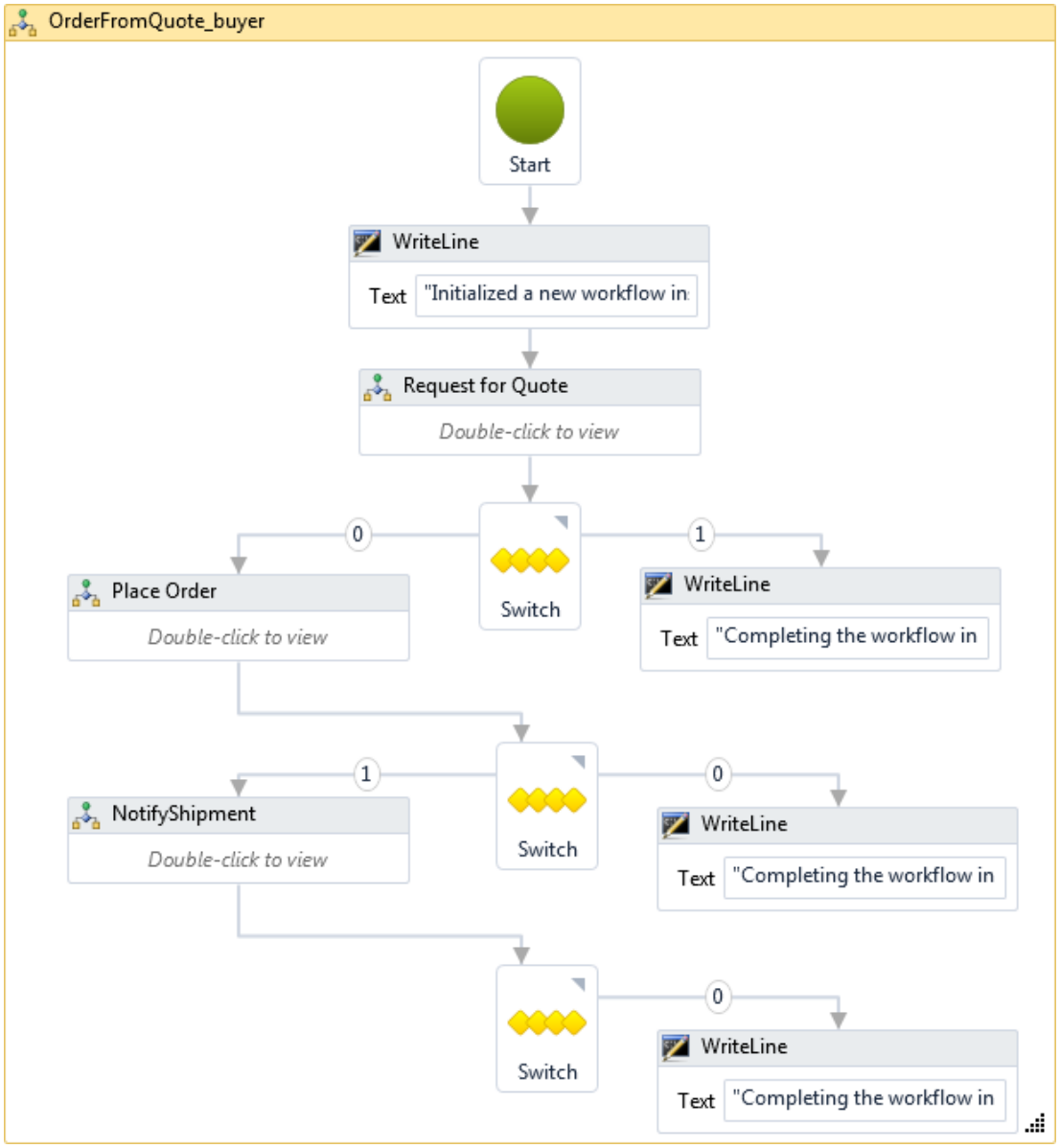

Abbildung 3.22: The workflow generated for the buyer in the "order from quote" example.

Variable Name	Type	Usage context
businessDecisionIndex	Int32	Holds the integer encoded business decision of a user based on an incoming request and is used to decide which reply to send back to the requesting party.
collaborationId	System.Guid	Holds the unique id used for correlating all messages to one business collaboration. This id is also assigned to each outgoing message so correlation will work.
contentHandle	CorrelationHandle	A handle which is necessary for content based correlation.
lastTransactionOutcome	Int32	After having finished a *business transaction*, this variable holds the decision for the switch activity, which comes next in the control flow to implement the subsequent branching behavior.
var1,var2,...,varN	IdContainer<T>	Holds each sent or received message with the type parameter "T" being an actual business document representing type or an acknowledgment type. *IdContainer* represents a wrapper type, which comes with an extra *Guid* used for content based correlation.
skipFirstReceiveInTransaction	Boolean	Only used inside service workflows. There, the first Receive activity from the first *business transaction* has been dragged out to enable this scenario. The first *business transaction* in such workflows has to skip the receiving of a business document, which is indicated by the value of this variable. The reason for this move is explained in the section "transformation step three: postprocessing".

Tabelle 3.1: Global variables defined in generated workflow definitions

Variable Name	Type	Usage context
retryCount	Int32	"The requesting authorized role must re-initiate the *business transaction* so many times as specified by the retry count in case that a time-out-exception – by exceeding the time to acknowledge receipt, or the time to acknowledge processing, or the time to respond – is signaled. This parameter only applies to time-out signals and not document content exceptions or sequence validation exceptions – i.e., failed business control exceptions." [16]
experiencedTimeout	Boolean	Holds whether a timeout has been experienced while waiting for an incoming message which influences the control flow.
resultingEntityStates	String[]	Holds possible outcomes for a *business transaction*. As an example those would be "quote [provided]" or "quote [refused]" for the "request for quote" *business transaction*.
replyOptions	List<Object>	Used only by responding parties to hold the available responding messages subsequently used for communicating a business decision to the business partner.

Tabelle 3.2: Local variables defined in generated *business transaction* definitions

and realize the business application communication that way. While possible, this approach comes with high costs as the setup on both sides must be synchronized to work together even though the hosted workflow and the business application are executing within the same process. An easier way to achieve the same result is to provide specialized custom activities within a custom .NET assembly which is needed anyways to transport specialized types such as the IdContainer<T> type described in table 3.1. Figure 3.23 shows the types provided by the Tuwien.Big.Bsopt.Workflow assembly written to support any BSopt derived workflow infrastructure. Right on top of the illustration there are three different classes deriving from the *NativeActivity* type. These are custom activities used for the following reasons:

- The *BusinessDecisionActivity* type takes a list of objects corresponding to possible responding messages and returns the zero based index of the message chosen from this list inside the business application. It provides a static C# event [38] which can be used by the business application to provide users with an opportunity to make business decisions.
- The *TransactionOutcomeDecisionActivity* type is used for mapping the information exchanged inside a *business transaction* to an actual outcome. It provides a list of possible *business transaction* outcomes and returns a zero based index defining the chosen result. Again it provides a static C# event which can be handled by a business application hosting the workflow.
- The *ExposeDataToHostActivity* type is an activity to provide generic data to event

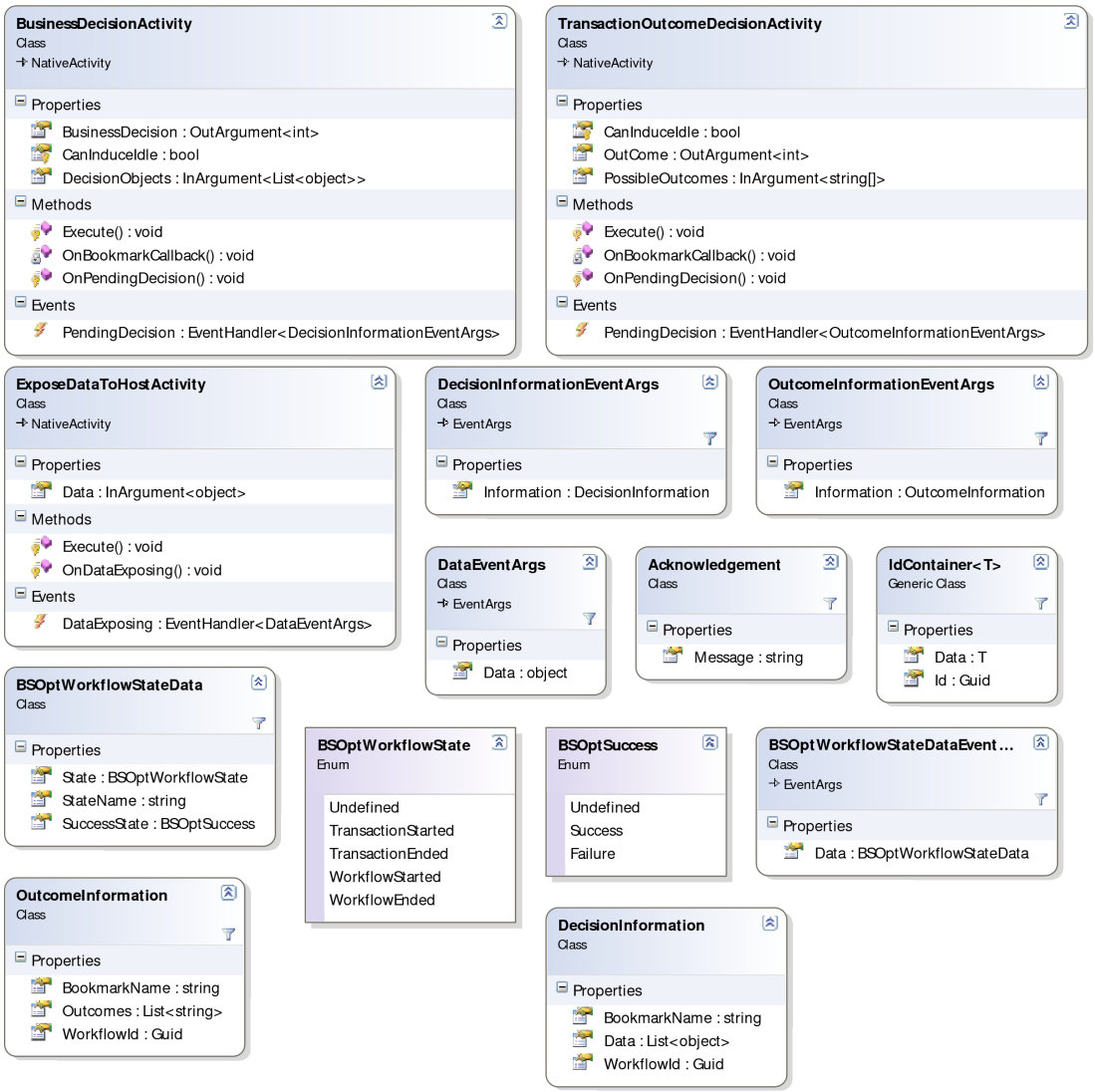

Abbildung 3.23: The types provided to any workflow definition by the Tuwien.Big.Bsopt.Workflow assembly.

consumers who were registering to its static *DataExposing* event. Its usage for BSopt workflows condensed down to notifying interested parties about the current state of the workflow which is realized by exposing *BSOptWorkflowStateData* instances through the event and handing over any received messages.

With the used variables and custom activities explained it's now possible to get back to the actual transformation steps and summarize what has to be done in order to generate a working *business transaction* representing nested flowchart activity. As activities inside a local choreography are independent from each other, it's possible to transform each element into new independent entities for the workflow definition. Table 3.3 shows the choreography activities involved in this process and their results within the workflow. The transformation of document exchanging interaction activities is also dependent on the given additional *userinput.xml* based meta data. It defines how many document possibilities are available. Also it sets whether resulting Receive activities shall be able to create new workflow instances.

Another specialty of message exchange activities is their need for correlation. A work-

Source Type	Transformation result
InitialActivity	An *ExposeDataToHostActivity* signaling the start of a new *business transaction*.
FinalActivity	In case of a business success a *TransactionOutcomeDecisionActivity* prompting the business application to decide on one business outcome used for further dispatching. Always followed by an *ExposeDataToHostActivity* signaling the end of a *business transaction*.
ValidationActivity	Validation types other than "check retry count" are covered by the validation functionality of the WCF runtime and need no further implementation inside the workflow. Else the value of the *retryCount* workflow variable is tested for being greater than zero and used for adequate branching.
InteractionActivity (activity type: business application)	For responding *business transactions* a *BusinessDecisionActivity* is emitted so the business application may prompt users for a decision based on the received business document information.
InteractionActivity (activity type: non-business application)	For outgoing activities a *Send activity* will be created. In case of multiple sending possibilities a *Switch activity* will decide which Send to use at runtime based on a previous business decision. For ingoing activities a *Picker activity* with as many *Receive activity* triggers as necessary and an additional *Delay activity* trigger used to react to timeouts will be created as is illustrated in figure 3.24. For received messages the custom *ExposeDataToHostActivity* type is used to signal the received message to the business application. In the case of a timeout the *experiencedTimeout* variable is set to true.

Tabelle 3.3: Basic transformation strategy for creating flowchart nodes from activities inside a local choreography

flow host may at every point in time manage multiple workflow instances possibly waiting for the same incoming message types. Hence, it's important to have a way to decide which received message belongs to which workflow instance. The chosen strategy to solve this problem is called "content based correlation". It decides the routing of messages based on their included information. As the messages exchanged by business partners are of a form not previously known to BSopt developers, the strategy for introducing content based correlation starts with wrapping each sent or received message inside the generic *IdContainer*<T> type shown in the class diagram in figure 3.23. This type is able to transport each message as its *Data* property and includes an additional *Id* property of type *System.Guid* which will act as a unique id used for the entire business collaboration. Content based correlation involves this type, all *Send* or *Receive* activities and the *correlation handle* variable *contentHandle* introduced in table 3.1. The *contentHandle* variable has to be initialized exactly once with a "query correlation initializer", which defines what to look for inside a given message. Once the variable has been initialized

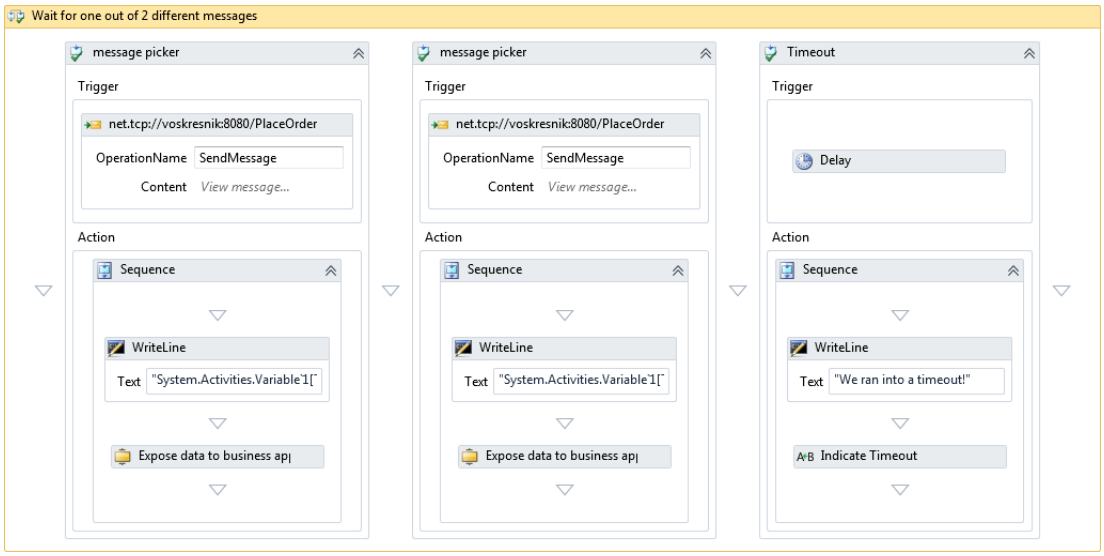

Abbildung 3.24: A pick branch including two *Receive activities* waiting for different incoming messages and a *Delay activity* to indicate a timeout.

it can be reused for finding correlation matches as often as desired. The default implementation for correlation queries is based on XPath expressions which have to be set up to match the Id property of each processed *IdContainer*<T> based message wrapper. In the used implementation the XPath expression "sm:body()/*/xg0:Id" is used with "sm:body()" being a custom XPath function provided by WCF. It finds the start of the actual message body inside a SOAP envelope. The transformer has to distinguish whether the currently processed *Interaction* activity is the first in execution order or not. If it is, the resulting WF messaging activity will be set up to initialize the *contentHandle* variable, else the activity will correlate based on the already initialized correlation handle.

Transformation step three: postprocessing

Phase 3 in the transformation process, the postprocessing step, is necessary to enable workflow definitions which result in the host creating new instances when receiving a first incoming message - so called "service workflows". Service workflows demand that they start with a *Receive* activity with its *CanCreateInstance* property explicitly set to `true`. Until this point it's not guaranteed that this is the case. The postprocessing step clones the *Receive activity* from the first *business transaction* in a service workflow definition. It inserts this cloned activity at the very start of the workflow to support the automatic creation of new workflow service instances. Also, it ensures that the activity's *CanCreateInstance* property is `true`. Then it adds a following global *Assign* activity to save the collaboration id taken from the received message into the global *collaborationId* variable. This id-value is used for the rest of the worfklow for setting up messages about to be sent. Only service workflows will depend on a collaboration id initialized externally. Non-service workflows on the other hand are starting with their *collaborationId* variable set to the default value of *Guid.NewGuid()* which is automatically creating a new unique id for the business collaboration. Finally it makes sure that the original *Receive* activity within the first *business transaction* will be skipped once so to prevent the workflow from trying to receive the same message type twice. It adds and uses the

```
<Activity
xmlns:sxs="clr-namespace:System.Xml.Serialization;assembly=System.Xml, Version
    =4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
xmlns:xsl="clr-namespace:Xml.Schema.Linq;assembly=Xml.Schema.Linq, Version
    =1.0.0.0, Culture=neutral, PublicKeyToken=null"
>
```

Listing 3.2: The two missing CLR namespace declarations necessary when including *LINQ to XSD* based business document classes in a workflow definition

global *skipFirstReceiveInTransaction* variable for this purpose which is set to `true` by default and will be set to `false` after the Receive activity has been skipped.

Transformation step four: serialization

Step 4 in the transformation process is to serialize the workflow instance that's been built on an object level into its XAML structure. It was also deemed desirable to be able to visualize the workflow as shown in figure 3.22, which is a screenshot taken from within Visual Studio 2010. The XAML specification is not only used to describe workflows but can be seen more generally as a technology to work with runtime object hierarchies. Subsequently the easiest way to serialize a workflow is to just use the static *Save()* method provided by the *System.Xaml.XamlServices* class and provide it with the workflow object and a filename. In [67] it is made clear that for workflow designers[5] to be able to visualize a XAML workflow, it must actually describe a type instead of an object hierarchy. This is supported by adding the *x:Class* attribute to the XML definition. The specialized *System.Activities.XamlIntegration.ActivityXamlServices* class provides a method for creating *XamlWriter* instances which support this scenario.

Unfortunately, tests showed that this is not enough. BSopt Designer generated workflows reference business documents which themselves depend on a common base class provided by the *LINQ to XSD* project (compare figure 3.17). The given *XamlWriter* instances will not detect this dependency and thus create results which can't execute or be visualized. The missing types had to be included manually: for CLR assemblies to be referenced inside a XAML instance the concept of XML namespaces has been extended. XML namespace defining URIs still can be mapped to prefixes which may then be used within the rest of the XML structure to reference elements residing in the specified namespace. As an example this is done with the declaration *xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"*, which is declaring an "x" prefix. Additionally, prefixes may be mapped to assembly specific .NET namespaces so managed types may be instantiated within a XAML definition. This is realized by supporting a special syntax to reference .NET namespaces as is documented in [45][6]. Listing 3.2 shows the two missing mapped CLR namespaces. After having been added to the original serialization results, these made the workflow infrastructure load the declared assemblies. It is noteworthy that the defined prefixes were never directly used inside the workflow definition. The automation of this process consists of two phases: first all necessary namespace declarations have to be detected for a given workflow definition. The resulting collection of namespace declarations then has to be written out instead of the namespaces the default implementation of *XamlXmlWriter* would serialize. The implementation thus reflects over all types it encounters referenced by a workflow definition

[5] In this context a 'designer' is just API-lingo for a visual editor.

[6] While this referenced documentation refers to WPF it's also applicable to WF.

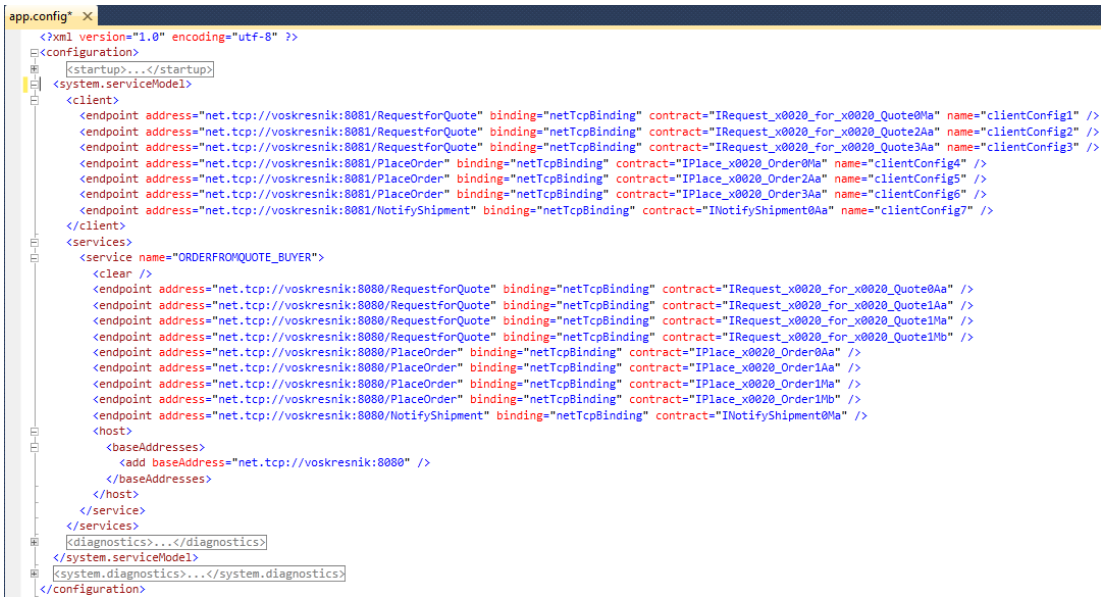

Abbildung 3.25: The generated application configuration file for the buyer for the "order from quote" example scenario.

```
52      </client>
53      <services>
54        <service name="<#=input.ServiceName#>">
55          <clear />
56  <#
57      string serviceEpTemplate =
58      @"          <endpoint address=""{0}"" binding=""{1}"" contract=""{2}"" />";
59
60      foreach(var sd in input.Services)
61      {
62          string endpointSection = string.Format(System.Globalization.CultureInfo.InvariantCulture,
63              serviceEpTemplate, sd.Address, input.BindingConfigElement, sd.ContractName);
64          WriteLine(endpointSection);
65      }
66  #>
```

Abbildung 3.26: An excerpt of the preprocessed app.config generating transformation template.

and creates distinct CLR namespace declarations for all types in their inheritance chain. It considers all implemented interfaces and also recursively checks all type arguments found in any generic type. Next a subclassed version of *XamlXmlWriter* is used to override specific methods such as *WriteNamespace* in order to add the additional namespace declarations which are given to the instance as additional constructor argument. With these extra efforts in place, the local choreography transformer supports the creation of workflows with external dependencies automatically considered and solved.

Generation of the application configuration file

The app.config file is a general concept predominantly used in .NET. It allows the declarative configuration of applications with user- or application- specific properties especially set up to be modifiable at deployment time. Additionally, if available, code can access specialized sections of the app.config file and read data from there. One example is the WCF, which allows the declaration of specific services using an app.config

```
void GenerateAppConfig()
{
    AppConfigTemplate appConfigTemplate = new AppConfigTemplate
    {
        InputFileName = _userInputFilePath
    };
    string genData = appConfigTemplate.TransformText();
    if (appConfigTemplate.Errors.Count > 0)
        throw new ArgumentException(string.Format(CultureInfo.CurrentCulture, "
            Transforming the application configuration file failed with {0}
            compiler errors.", appConfigTemplate.Errors.Count));

    using(var fs = File.Open(AppConfigFilename, FileMode.Create, FileAccess.
        Write, FileShare.None))
    {
        TextWriter tw = new StreamWriter(fs);
        tw.Write(genData);
        tw.Flush();
    }
}
```

Listing 3.3: Code to transform text with a preprocessed T4 template class

file instead of creating the desired setup in code. Using application files for service configuration at deployment time is desirable for BSopt Designer created workflows. The reason for this is, that it serves to decouple workflows from business applications. This is contrary to the approach of hard-coding the WCF objects hosted workflows require at runtime. App.config files can thus be used more universally as consuming business applications do not need to know in advance how to set up any services a hosted workflow might need. Figure 3.25 depicts an excerpt of the application configuration file generated for the buyer role for the "order from quote" example scenario. The XML block WCF is concerned with starts with the *system.servicemodel* element and contains client and service specific configurations including endpoint adresses, bindings, contracts and names. While the illustration might be quite self explanatory a thorough documentation of the related configuration XML schema is available at [43]. The structure of the app.config file supporting business applications is predefined with client and service elements repeating dynamically based on the information workflows want to exchange. This precondition is a perfect usage scenario for a preprocessed T4 text template which was already mentioned in section 2.5 on page 25.

The template contains a mixture of (i) static XML data and specific directives to either (ii) execute some code or (iii) write the result of a code evaluation into the output. Figure 3.26 shows an excerpt of the text template illustrating those three different kinds of data. Lines 52 to 55 show static output with the plotting of an evaluated code statement into the output at line 54. Finally, lines 56 to 66 feature C# code which has to be executed in order to generate the various endpoint elements seen in figure 3.25. The input variable which is referred to in the illustration is defined at the end of the text template and of kind *UserInput* as defined in figure 3.20. The text template is automatically transformed into a C# class which then can be used at runtime to transform data from a userinput.xml file into an app.config file. Listing 3.3 shows how the resulting *AppConfigTemplate* class is used by calling its *TransformText()* method and writing the returned string data to the file path given by the *AppConfigFilename* property.

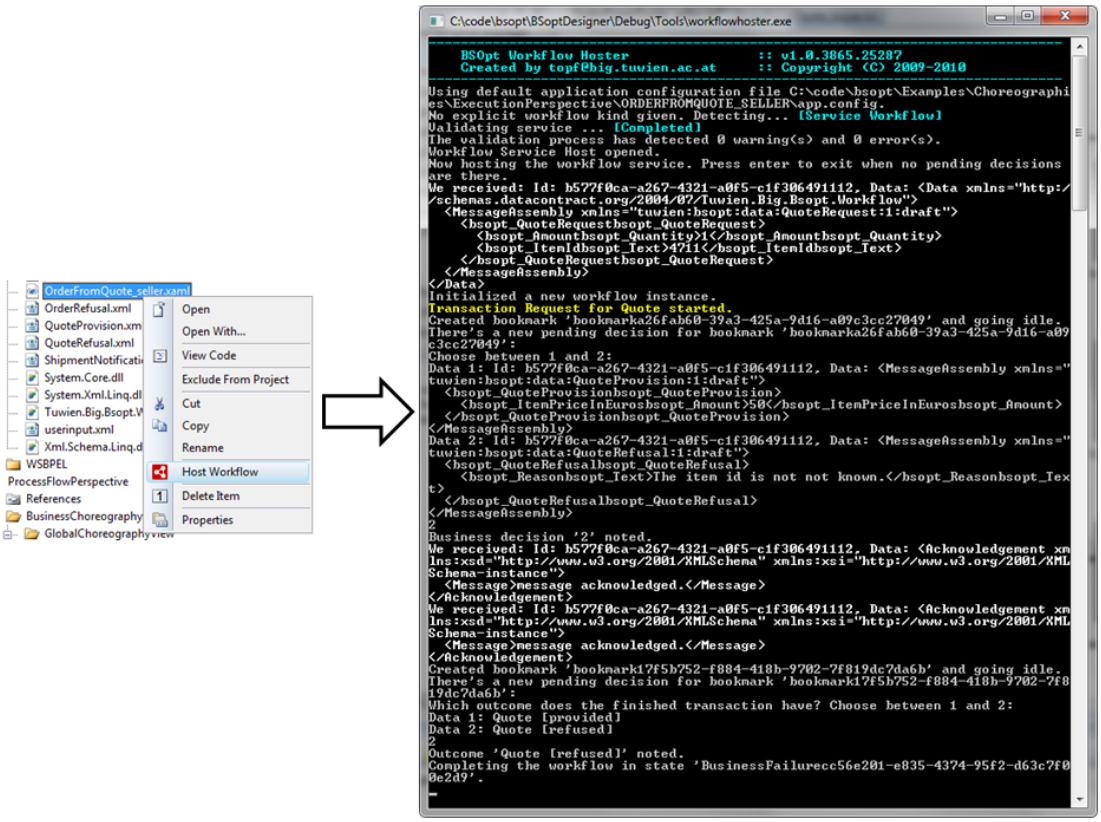

Abbildung 3.27: Hosting a generated WF4 workflow from within BSopt Designer.

3.5 The workflow hosting application

Workflows as definition of an executable business process have some merit in the visualization of their structure. However, their main benefit is that they enable the concise description of machine executable business logic. This allows prepared applications to consume and "just run" the workflow. The assemblies which come with BSopt Designer allow third party developers to create their own business applications. These can consume created workflow artifacts as all types needed - such as the custom activities described in the last section - are provided publicly. For testing reasons and for validating whether the generated process descriptions worked in the first place, BSopt Designer also provides users with a sample business application. It comes as a console application, automatically detects whether a given workflow is a service- or non-service workflow and hosts the given business logic accordingly. This section will describe this hosting application by first presenting the user perspective and then describing the application's implementation. As the hosting application's behavior is very much depending on the hosted workflow definition, a specific use-case description with predefined processing structure is omitted. However, the ways users can interact within the bounds of the business process will be described in the following.

In order to launch the workflow hosting application, an *execution perspective project* must be opened in BSopt Designer's solution explorer. This project type offers an additional context menu for files with the file extension .xaml named "Host Workflow". Users selecting this menu entry will launch the BSopt sample workflow host for the selected file as illustrated in figure 3.27. This console application will by default look for an app.config file within the .xaml file's directory on the filesystem and use this for set-

ting up its client- and service endpoints. Next - if not told specifically - the application will try to detect whether it's dealing with a service- or non-service workflow. Finally, the workflow gets validated so any configuration warnings or errors will be detected before hosting starts. Based on the detected or given kind of workflow, the hoster will either wait for specific incoming messages which can trigger the instantiation of a new workflow or launch one non-service workflow instance immediately.

The illustration in figure 3.27 shows the hosting of artifacts created for the seller for the "order from quote" example scenario. Subsequently, the hosting business application detects a service workflow as the seller's role is to wait for incoming *QuoteRequest* messages. As shown in the screenshot, the application receives a quote request from a buyer asking for a quote about one item with id 4711. This leads to the instantiation and execution of a new workflow and toward a business decision requiring human interaction: should the seller provide a quote or refuse it? For brevity the user in this scenario chooses to refuse the quote. This makes the business application transcend into the "Quote [refused]" state. Thus, the execution logic of the workflow reaches an end. It recognizes a business failure for this example. The buyer's side reaches the same conclusion after receiving the seller's reply. According to the underlying information originating from the global choreography, the host of the buyer-workflow also recognizes the end of the business process which leads to a business failure end-state.

Had the seller instead provided a quote by choosing to reply with a *quote provision message*, the workflow would have completed the "Request for Quote" business transaction with a business success state. This would have lead the control flow into the "Place Order" business transaction, where the same basic scenario would then repeat. In the case of a successful order, the final business transaction in the "order from quote" example scenario is called "notify shipment". Here the seller has the role of the initiant and the buyer is defined not to reply. Hence, the very last message of a successful business process execution is the seller notifying the buyer about the shipping of the ordered goods.

Implementation of the workflow hosting application

The workflow hosting application is based on a main executable which is depending on types from the *Tuwien.Big.Bsopt.Workflow* assembly. Both components themselves rely on additional external assemblies provided by the .NET framework. The implementation of the workflow hosting application is separated into four stages:

1. Parsing of commandline arguments and workflow host initialization.
2. Optional detection of the workflow kind.
3. Hosting of the workflow based on its kind.
4. Handling of workflow events.

In the following each of those stage will be discussed.

1. Parsing of commandline arguments and workflow host initialization

The workflow host offers commandline options to specify a custom application configuration file which shall be used for the set up of WCF based endpoints and to explicitly define the workflow kind. In most cases the application may be left to find the most appropriate setting on its own. Application configuration files are by default associated with applications when their filename is the concatenation of the source application's

filename with the string ".config". For the workflow hosting application "workflowhoster.exe" the application configuration file name would thus be named "workflowhoster.exe.config". It's clearly undesirable to require workflow supporting application configuration files to be copied into the workflow host application's directory and to be renamed to a specific filename, possibly overwriting other files in this process. The hosting application thus solves this problem by using the concept of *application domains ("appdomains")*: *"Application domains provide an isolation boundary for security, reliability, and versioning, and for unloading assemblies."* [37] Another feature they support is the usage of custom application configuration files which makes it possible to specify a selected app.config file at runtime instead of deployment time. In order to use the custom configuration file the workflowhost thus creates a new *appdomain*, creates and unwraps the main hosting class in it and proceeds to control this instance from the original *appdomain*. Types in different *appdomains* are completely isolated from each other. Yet it's still possible to communicate with code in another appdomain by using .NET remoting features and either make sure the data transferred into another appdomain is serializable or supports the creation of proxy objects e.g. by deriving from the special *System.MarshalByRef* type. The hosting application defines a *WorkflowHoster* class which does the latter and provides an *Execute* method which is called when its initialization is complete. The initialization in the class constructor consists of loading all assemblies in the destination workflow's directory to make sure all types referenced later can be resolved correctly and by collecting additional configuration data sent from the original *appdomain* scope of the application.

2. Detection of the workflow kind

The detection of the workflow kind is straightforward. It uses a *XamlXmlReader* instance to iterate over all XAML elements in the workflow and looks whether it can detect a *CanCreateInstance* property which is only set explicitly when its default value of false has been changed. In case this property has been detected the workflow is deemed to be a service workflow, else it's of the non-service workflow kind.

3. Hosting of the workflow

The hosting of the workflow service is based on the kind of workflow detected or explicitly given. What's the same for both kinds is the creation of a *WorkflowService* instance from the given workflow definition, which can be validated using the *System.Activities.Validation.ActivityValidationServices* class and handed over to a *WorkflowServiceHost* instance, which is responsible for hosting workflows and support the messaging infrastructure. This instance will start to host the workflow service given to it when its *Open* method has been called. It will also create all endpoints specified via the custom application configuration file linked with the application domain in which the host is executing.

For non-service workflows an instance has to be launched explicitly, which is achieved by communicating with the workflow runtime using a special workflow management endpoint. It has to be added to the workflow service host before it can be accessed using a *WorkflowControlClient* instance. This class is able to control already existing workflow instances and is used in the application to unsuspend the freshly created non-service workflow. As specified in the WF documentation in the section "Workflow Service Host Extensibility" the only way to launch a non-service workflow is to derive from the *WorkflowHostingEndpoint* class. One has to override the methods linked to instance creation to work together with a custom service contract interface defined to

```
public class BusinessDecisionActivity : NativeActivity
{
    public static event EventHandler<DecisionInformationEventArgs> PendingDecision;

    public OutArgument<int> BusinessDecision { get; set; }
    public InArgument<List<object>> DecisionObjects { get; set; }

    protected override bool CanInduceIdle
    {
        get { return true; }
    }

    protected override void Execute(NativeActivityContext context)
    {
        //create a bookmark
        string bookMarkName = "bookmark" + Guid.NewGuid().ToString();
        context.CreateBookmark(bookMarkName, new BookmarkCallback(OnBookmarkCallback));
        Console.WriteLine("Created bookmark '{0}' and going idle.", bookMarkName);

        OnPendingDecision(bookMarkName, context.WorkflowInstanceId, DecisionObjects.Get(context));
    }

    void OnBookmarkCallback(NativeActivityContext context, Bookmark bookmark, object state)
    {
        //write a message when bookmark resumed
        int decision = int.Parse((string)state);
        BusinessDecision.Set(context, decision);
        Console.WriteLine("Business decision '"+(decision+1)+"' noted.");
    }

    protected virtual void OnPendingDecision(string bookmarkName, Guid workflowInstanceId, List<object> decisionObjects)
    {
        DecisionInformation info = new DecisionInformation
        {
            BookmarkName = bookmarkName,
            WorkflowId = workflowInstanceId,
            Data = decisionObjects
        };

        if (null != PendingDecision)
            PendingDecision(this, new DecisionInformationEventArgs(info));
    }
}
```

Abbildung 3.28: The source code of the BusinessDecisionActivity written to let users input their business decision into a workflow.

create a new workflow instance. After this endpoint has been added to the workflow host, the service contract interface may be used in conjunction with a WCF channel factory to create a proxy object. This object allows to issue a workflow instance creation command to the (not directly exposed) workflow runtime.

The last piece of code in the hosting process is an additional thread, which is used to react to user input and either close the application or accept data entry for pending decisions.

4. Handling of workflow events

While the previous descriptions are sufficient to support the hosting of generic workflows, the workflow hoster at hand has to be more specialized in order to support the scenarios common to all created workflow instances. As presented in the last section BSopt Designer generated workflows consist of custom activities used to provide users with the opportunity to make business decisions or to exchange states or received data with a business application. The mechanism to influence workflows from outside and exchange data is enabled by so called *bookmarks*. Those are named entities which support the resumption of a workflow with a linked callback method. By default custom activities which create *bookmarks* while executing are not considered completed until the *bookmark* itself has been resumed. Code outside the workflow can resume a pending *bookmark* if it knows the *bookmark* name and also transfer data back to the workflow at

the same time. This is used by the custom activities to receive user decisions and by the business application to be able to influence running workflows. Figure 3.28 shows the entire source code of the *BusinessDecisionActivity* which lets users input their decisions into a running workflow instance by providing users with a set of decision objects to choose from. It consists of an Execute method which is called by the workflow runtime when the activity shall execute. Inside the method a *bookmark* with a unique name is created which, when resumed will lead to the execution of the *OnBookmarkCallback* method. Next it will raise its static *PendingDecision* event which will transport the *bookmark* name, the workfklow instance id and all decision objects to any code which registered for the event. When the *bookmark* is resumed by external code the *OnBookmarkCallback* method interprets the returned state as the chosen index into the list of decision objects. This in turn can be used by other activities inside the workflow to drive its execution forward. The inner workings of the two other custom activities are based on the same principle. The workflow hosting application, by registering to the events exposed by all three custom activities is provided with the relevant information from a workflow instance. Using this data it can resume the workflow at hand as it was provided with the right *bookmark* name as part of each event related data.

4 Conclusion and outlook

This thesis covered the transformation of logical business process descriptions into executable workflow artifacts and their subsequent deployment in a custom example business application. The process is aligned with the two main views described in the Openedi reference model [20]. The approach for BSopt [2] identifies three distinct perspectives which influence B2B collaborations (figure 1.1): The value perspective considers the economic drivers for the realization of business models as seen by management. In the process flow perspective business analysts direct the formulation of logical business process descriptions based on given business models. Finally, the execution perspective is managed by IT specialists who convert business process descriptions into executable workflow artifacts ready for integration into their service oriented architecture. The BSopt approach aims to integrate these three differing layers by providing domain specific languages integrated into the tool environment called "BSopt Designer" [24]. It provides a semi automatic, wizard guided mapping between the different areas in order to efficiently transport information from one domain into the other. This strategy helps to shorten turn-around times as given information need not be reinterpreted by humans.

This thesis described four main contributions supporting the aforementioned BSopt approach. They follow the contributions as given in section 1.2 on page 4:

- A wizard was integrated into BSopt Designer to support the transformation of a global choreography given in the form of the BCL [47] into local choreographies defined as LCL models.
- Another wizard was created to transform message type descriptions given as XML schemas into CLR compatible types. The type definitions are then compiled into a .NET assembly to support the messaging logic inside workflow artifacts.
- A third wizard was built to map an LCL model description and a compatible message type assembly into WF4 based workflow artifacts.
- An example business application was written which allows the automatic one-click hosting of generated workflow artifacts from within BSopt Designer.

The presentation of the different transformation implementations is accompanied by an example business process. It describes a buyer communicating with a seller in order to receive a quote for an item. Next the buyer continues to order the specified item and finally the seller notifies the buyer when the ordered goods have been shipped. This example comes as a business process description elaborated from a neutral perspective. It gets transformed into two local choreography descriptions showing the process from the perspective of each participant. Next, these process descriptions result in two complementary XAML based workflow artifacts which are ready to communicate with each other in a fashion based on the settings entered in the workflow generation wizard. The hosting of both workflows in the workflow hosting application finally proves both their validity and correct functionality. It shows, that the generated workflows change their execution flow as expected based on the decisions users make. The environment must only support a small set of requirements necessary to drive any business process which

demands human interaction. Hence, the workflow hoster is able to support a wide range of different BSopt Designer generated workflow artifacts.

Despite the fine results obtained when validating the implementation, there are still open research areas demanding attention in the future. First, the concepts of compensation and extended control flow constructs are not yet considered in the transformation of global choreographies. Moreover, business process descriptions would also profit from the integration of evaluation statements which could facilitate tool based automatic decision making, e.g., to accept only goods no more expensive than a given baseline value. Finally, the current workflow artifacts come with no generated layout which handicaps their visualization for developers. A natural step possible to improve the system would be to change this fact and introduce a fitting automatic layout such as was done for the local choreography DSL model.

Concluding, the transformation of business process descriptions into workflow artifacts has demonstrated real usage value. We compared it with the process necessary to create equivalent workflows from scratch. The modeling approach in BSopt Designer showed, that it is possible to have an example process such as the one presented in this thesis up and running within minutes. The manual realization of equivalent workflows took much longer to develop and due to the required complex combination of properties left much room for accidental errors. It is also easy to adapt processes to changing economic realities as new models on higher levels can be transformed back into workflow artifacts at any time.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] Aalst, V. D. The application of petri nets to workflow management, 1998.

[2] BSopt. Business semantics on top of process technology. Project website online at http://www.bsopt.at.

[3] Chappell, D. The workflow way - understanding windows workflow foundation. Online at http://www.davidchappell.com/TheWorkflowWay–Chappell.pdf, April 2009.

[4] ECMA. Standard ecma-335 common language infrastructure (cli) 4th edition. Online available at http://www.ecma-international.org/publications/standards/Ecma-335.htm, June 2006.

[5] Flanders, J. Windows workflow foundation integration with windows communication foundation. Online at http://msdn.microsoft.com/en-us/library/cc626077.aspx, May 2008.

[6] Fowler, M. Language workbenches: The killer-app for domain specific languages? Online at http://martinfowler.com/articles/languageWorkbench.html, June 2005.

[7] Fowler, M. *Domain Specific Languages*. Addison-Wesley Professional, 2010. final draft available at http://my.safaribooksonline.com/9780132107549.

[8] Fowler, M., and Evans, E. Fluentinterface. Online at http://www.martinfowler.com/bliki/FluentInterface.html, December 2005.

[9] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mef, R., and Stafford, R. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[10] Free Software Foundation, I. GNU Gereral Public License, Version 3. Online at http://www.gnu.org/licenses/gpl-3.0.html, June 2007.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[12] Georgakopoulos, D., Hornick, M., and Sheth, A. An overview of workflow management: From process modeling to workflow automation infrastructure. In *DISTRIBUTED AND PARALLEL DATABASES* (1995), pp. 119–153.

[13] Hejlsberg, A., and Torgersen, M. Overview of C# 3.0. Online at http://msdn.microsoft.com/en-us/library/bb308966.aspx, March 2007.

[14] Hofreiter, B. Registering uml models for global and local choreographies. In *ICEC '08: Proceedings of the 10th international conference on Electronic commerce* (New York, NY, USA, 2008), ACM, pp. 1–10.

[15] Hollingsworth, D. The workflow reference model (issue 1.1). Available from http://wfmc.org/reference-model.html, January 1995.

[16] Huemer, C., Dietrich, J., Hofreiter, B., Liegl, P., Miller, G., Moyer, H., Schuster, R., and Zapletal, M. UN/CEFACT's Modeling Methodology (UMM) Meta Model - Foundation Module Candidate for 2.0 Draft for IMPLEMENTATION VERIFICATION. Available online from http://www.untmg.org/specifications/, October 2009.

[17] Huemer, C., Liegl, P., Schuster, R., Werthner, H., and Zapletal, M. Inter-organizational systems: From business values over business processes to deployment. *Digital Ecosystems and Technologies, 2008. DEST 2008. 2nd IEEE International Conference on 1* (2008), 294–299.

[18] Huemer, C., Liegl, P., Schuster, R., and Zapletal, M. A 3-level e-business registry meta model. *Services Computing, IEEE International Conference on 1* (2008), 441–450.

[19] Huemer, C., and Zapletal, M. A State Machine executing UMM Business Transactions. In *Digital EcoSystems and Technologies Conference, 2007. DEST '07. Inaugural IEEE-IES* (21-23 2007), pp. 57 –62.

[20] ISO. Information technology - open-edi reference model, 2004. Second Edition.

[21] Kaplan, J., and Santos, L. F. Clr inside out - in-process side-by-side. MSDN Magazine December 2009 Issue, December 2009. Online available at http://msdn.microsoft.com/en-us/magazine/ee819091.aspx.

[22] Kumar, M. R. A simple wizard control for .net 2.0 with full designer support. Online at http://www.codeproject.com/KB/cs/WizardDemo.aspx, February 2008.

[23] Liegl, P., and Mayrhofer, D. A domain specific language for un/cefact's core components. *Services Part II, IEEE Congress on 0* (2009), 123–131.

[24] Liegl, P., Schuster, R., Zapletal, M., Motal, T., Mayrhofer, D., and Topf, M. BSopt Designer. Online, 2010. http://code.google.com/p/bsopt-designer/.

[25] Martin, R. C. *Clean Code - A handbook of agile software craftmanship*, vol. 5. Prentice Hall, 2009.

[26] Microsoft. Domain-specific language tools. Online at http://msdn.microsoft.com/en-us/library/bb126235

[27] Microsoft. Microsoft .NET Framework. Online at http://www.microsoft.com/net/. http://www.microsoft.com/net/.

[28] Microsoft. .NET Framework Conceptual Overview. Online at http://msdn.microsoft.com/en-us/library/zw4w595w

[29] Microsoft. Xml schema definition tool (xsd.exe). Online at http://msdn.microsoft.com/en-us/library/x6c1kb0s2005.

[30] Microsoft. [ms-xaml]: Xaml object mapping specification 2006 v1.0. Available online at http://msdn.microsoft.com/en-us/library/dd3618522006.

[31] Microsoft. Servicemodel metadata utility tool (svcutil.exe). Online at http://msdn.microsoft.com/en-us/library/aa3477332007.

[32] Microsoft. Windows Communication Foundation (.NET Framework 3.5 development reference). Online at http://msdn.microsoft.com/en-us/library/ms735119

[33] Microsoft. Windows Workflow Foundation (.NET Framework 3.5 development reference). Online at http://msdn.microsoft.com/en-us/library/ms735967

[34] Microsoft. Domain-specific language tools - generating artifacts by using text templates. Online at http://msdn.microsoft.com/en-us/library/bb126445

[35] Microsoft. Interoperability overview (c# programming guide). Online at http://msdn.microsoft.com/en-us/library/ms173185

[36] Microsoft. Visual studio automation and extensibility - the spectrum of visual studio automation. Online at http://msdn.microsoft.com/en-us/library/9b54865a

[37] Microsoft. Application domains. Online at http://msdn.microsoft.com/library/EN-US/113A8BBF-6875-4A72-A49D-CA2D92E19CC8(VS.100), 2010.

[38] Microsoft. C# language specification 4.0. Online available from http://www.microsoft.com/downloads/details.aspx?FamilyID=dfbf523c-f98c-4804-afbd-459e846b268e&displaylang=en, April 2010.

[39] Microsoft. Language-integrated query (linq) - linq to xml. Online at http://msdn.microsoft.com/library/EN-US/F0FE21E9-EE43-4A55-B91A-0800E5782C13(VS.100), 2010.

[40] Microsoft. Linq to xsd. Online at http://linqtoxsd.codeplex.com/, April 2010.

[41] Microsoft. [MC-NETCEX]: .NET Context Exchange Protocol Specification. Available online at http://msdn.microsoft.com/en-us/library/cc4419822010.

[42] Microsoft. [MS-XAML-2009]: XAML Object Mapping Specification 2009. Available Online at http://msdn.microsoft.com/en-us/library/ff6291552010.

[43] Microsoft. system.servicemodel. Online at http://msdn.microsoft.com/library/EN-US/78519531-AD7A-40D3-B3E7-42F1103D8854(VS.100), June 2010.

[44] Microsoft. Wf guidance: Workflow services. Available online at http://www.microsoft.com/downloads/details.aspx?FamilyID=bd94c260-b5e0-4d12-93ec-53567505e685&displaylang=en, 1 2010.

[45] Microsoft. Xaml namespaces and namespace mapping for wpf xaml. Online at http://msdn.microsoft.com/en-us/library/ms747086.aspx, 2010.

[46] Milner, M. Foundations: Workflow services. MSDN Magazine - Launch 2008 Issue Online at http://msdn.microsoft.com/en-us/magazine/cc164251.aspx, 2008.

[47] Motal, T., Zapletal, M., and Werthner, H. The business choreography language (bcl) - a domain-specific language for global choreographies. *Services Part II, IEEE Congress on 0* (2009), 150–159.

[48] OASIS. Web services reliable messaging tc ws-reliability 1.1 oasis standard. Online at http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf, November 2004.

[49] OASIS. Reference model for service oriented architecture. Available online at http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf, October 2006. OASIS Standard, Version 1.0.

[50] OASIS. Web services security: Soap message security 1.1 (ws-security 2004) oasis standard specification. Online at http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, February 2006.

[51] OMG. Documents associated with business process model and notation (bpmn) version 2.0 - beta 2. Online at http://www.omg.org/spec/BPMN/2.0/Beta2/, May 2010.

[52] (OMG), O. M. G. Business process modeling notation specification (bpmn), January 2009. Version 1.2.

[53] Pavlov, D. Visual studio managed package framework for projects (mpfproj). Online at http://mpfproj.codeplex.com/, 2008.

[54] Peltz, C. Web services orchestration and choreography. *Computer 36*, 10 (2003), 46–52.

[55] Schuster, R., and Motal, T. From e3-value to REA: Modeling Multi-party E-business Collaborations. *E-Commerce Technology, IEEE International Conference on 0* (2009), 202–208.

[56] Simpkins, C. The road to wf 4.0 (part 1). Online at http://blogs.msdn.com/b/endpoint/archive/2009/01/20/the-road-to-wf-4-0-part-1.aspx, January 2009.

[57] Sych, O. T4 Architecture. Online at http://www.olegsych.com/2008/05/t4-architecture/, May 2008.

[58] Sych, O. Understanding T4: Preprocessed Text Templates. Online at http://www.olegsych.com/2009/09/t4-preprocessed-text-templates/, 2009.

[59] Technical University of Dortmund, Friedrich-Schiller-University Jena, University of Cologne and oreas GmbH. Ogdf - open graph drawing framework. Online available at http://www.ogdf.net, December 2007.

[60] UN/CEFACT. Core components technical specification version 3.0. Available online at http://www.unece.org/cefact/codesfortrade/CCTS/CCTS-Version3.pdf, September 2009.

[61] van der Aalst, W. M., and Lassen, K. B. Translating unstructured workflow processes to readable bpel: Theory and implementation. *Information and Software Technology 50*, 3 (2008), 131 – 159.

[62] van der Aalst, W. M. P., Ter, Kiepuszewski, B., and Barros, A. P. Workflow patterns. *Distributed and Parallel Databases 14*, 1 (July 2003), 5–51.

[63] W3C. XML Path Language (XPath) Version 1.0. Online at http://www.w3.org/TR/xpath/, November 1999.

[64] W3C. Web Services Description Language (WSDL) 1.1. Online at http://www.w3.org/TR/wsdl, March 2001.

[65] Winkler, M. Advanced workflow services talk (demo 2 of 4). Online at http://blogs.msdn.com/b/mwinkle/archive/2008/08/06/advanced-workflow-services-talk-demo-2-of-4.aspx, Aguust
2008.

[66] Winkler, M. Q & a on advanced workflow services talk. Online at http://blogs.msdn.com/b/mwinkle/archive/2008/08/07/q-a-on-advanced-workflow-services-talk.aspx, August
2008.

[67] Winkler, M. Types, metatypes and bears, oh my! Online at http://blogs.msdn.com/b/mwinkle/archive/2009/06/10/types-metatypes-and-bears-oh-my.aspx, June
2009.

[68] zur Muehlen, M. *Workflow-based Process Controlling - Foundation, Design, and Application of Workflow-driven Process Information Systems.* Logos Verlag Berlin, 2004.