Bridging the gap between business process models and Service Oriented Architectures with reference to the Grid Environment

Zaheer Khan*
Centre for Complex Cooperative Systems, Dept. of CSCT, FET,
University of the West of England,
BS16 1QY, Bristol, United Kingdom
Email: Zaheer2.khan@uwe.ac.uk
* Corresponding Author

Mohammed Odeh
Centre for Complex Cooperative Systems, Dept. of CSCT, FET,
University of the West of England,
BS16 1QY, Bristol, United Kingdom
Email: Mohammed.Odeh@uwe.ac.uk

Richard McClatchey
Centre for Complex Cooperative Systems, Dept. of CSCT, FET,
University of the West of England,
BS16 1QY, Bristol, United Kingdom
Email: Richard.McClatchey@uwe.ac.uk

Abstract: In recent years organisations have been seeking technological solutions for enacting their business process models using ad hoc and heuristic approaches. However, limited results have been obtained due to the expansion of business processes across geographical boundaries and the absence of structured methods, frameworks and/or Information Technology (IT) infrastructures to enact these processes. In an attempt to enact business process models using distributed technologies, we introduce a novel architectural framework to bridge the gap between Business Process Models and Grid-aware Service Oriented Architectures (GSOA). BPMSOA framework is aligned with the Model Driven Engineering (MDE) approach and is instantiated for role-based business process models, (in particular Role Activity Diagramming (RAD)) using mobile process languages such as pi-ADL. The evaluation of the BPMSOA framework using the Submission process from the digital libraries domain has revealed that role-based business process models can be successfully enacted in GSOA environments with certain limitations.

Key words: Service Oriented Architecture, RAD business process models, Metamodels, pi-ADL, Process Enactment, Model Driven Engineering, Model Transformation

Bibliographic Notes: Dr. Zaheer Khan is a postdoctoral research fellow in the Faculty of Environment and Technology of UWE, Bristol, UK. He has over seven years of teaching and research experience in system analysis and design, databases, distributed systems, etc. He has been actively working in large scale EU and non-EU collaborative computing research projects: Health-e-Child, HUMBOLDT, LifeWatch, SAGE. His main research interests are in distributed computing, software engineering, business process management, data interoperability and management, software agents, and geospatial information systems.

Dr. Mohammed Odeh is a senior lecturer and leading the Software Engineering Research Group in the Centre for Complex Cooperative Systems of UWE, Bristol, UK. He has over twenty years of experience of working in industry and academia. His areas of interest are software and requirement engineering, intelligent systems, business process management.

Professor Richard McClatchey has been research active for the past 30 years and has led many projects funded by industry and by the EC in the areas of large-scale distributed data and process management and database modelling and in systems design and integration. Prof. McClatchey currently leads the Centre for Complex Cooperative Systems at UWE and is active in collaborative computing projects at CERN, and with many international partners in other EC projects: Health-e-Child, SHARE and neuGRID.

1. INTRODUCTION

Business process modelling is the process of visualising business processes in order to understand, improve and/or enact these processes in a particular computing environment. The availability of different business process modelling approaches (such as role-based modelling) and their associated process modelling languages (Giaglis, 2001; Curtis, 1992; Aguilar-Saven, 2003; Havey, 2005) enable organisations to capture and model different perspectives of their business processes. In the presence of such diverse process modelling approaches, the homogeneous enactment and execution of business processes is not straightforward. Furthermore, such enactment becomes more complicated when businesses expand across geographical boundaries. Thus, it becomes challenging to find solutions from Technological Spaces (TS) (Kurtev, et. al, 2002) in the absence of homogeneous and structured methods or frameworks that can provide mechanisms to enact these business processes using distributed technologies.

It is envisaged that environments that adopt the Service Oriented Architecture (SOA) (Krafzig, et. al, 2004) model of computing are increasingly becoming the *de facto* environment for executing business processes represented by their visual models (BEA Systems, 2007). Service oriented computing can utilise the characteristics of both Object-oriented and Component-based (Breivold, et. al, 2007) computing models in an open environment. However, the enactment and execution of business process models in service-oriented environments is not straightforward since it requires appropriate mappings and translations of business process models into system models for their execution in such environments, with the objective of utilising application domain specific services and also for facilitating business-to-business (B2B) process integration.

In this context, the research here presented is an attempt to bridge the gap between business process models and service oriented environments (such as the grid-aware SOA (GSOA) environment). This gap can be mainly attributed to the following challenges:

a. As there exist several business process modelling approaches (e.g. role-based, functional, activity-based, etc) and their associated business process modelling languages (e.g. Role Activity Diagrams (Ould, 1995), Business Process Modelling Notations (BPMN 2007), UML Activity Diagrams, etc) with different syntactic and semantic notations, a structured approach is needed to uniformly handle the instantiation and execution (i.e. enactment) of business process models;
b. As business process models are mainly intended for domain experts and system models are for IT experts, an autonomy between business process models and system models should be maintained;
c. Not all graph-based visual process models can be directly translated into block-structured models due to conceptual differences between graph-based and block-structured process models which restrict one-to-one mappings and translations from former to later; and
d. Reusable transformation of visual business process models into executable workflow specifications and/or code that can be instantiated in distributed environments and in particular GSOA.

As a first step in bridging the gap between business process models and GSOA, application domain specific services (e.g. utility services) can be deployed on top of grid core services in order to access grid resources by encapsulating the complexity of low level grid-specific tasks and facilitating business applications for executing their business processes as envisioned in (NGG, 2006; SOKU, 2006). Such application domain specific services can be referred to as grid-aware services or utility services. For example, Web Service Resource Framework (WSRF) based application services are capable of accessing grid infrastructure services (Grid Resource Allocation Manager - GRAM, Monitoring and Discovery Service – MDS, etc.) and resources when deployed using the Globus Toolkit 4 (Sotomayor, 2005). As a second step business process models would then require rigorous mappings and translations into the source code of executable programming models and/or languages capable of accessing grid resources. Such an approach necessitates homogeneous structured methods and tools to avoid complexity in performing the aforementioned translations.

In the above context, this paper presents a novel approach in bridging the gap between business process models and grid-aware service oriented environments. It introduces an architectural framework, namely the BPMSOA (*Business Process Models for Service Oriented Architectures*) framework (Khan 2009), which adapts Model Driven Engineering (MDE) (Stuart, 2002; Fondement, et. al, 2004) approach to transform visual process models (irrespective of business process modelling approach and/or modelling language) into concrete code that is executable in a GSOA environment. This approach not only maintains autonomy between business process models and system models but also promotes reusability of translated artefacts. As part of the BPMSOA framework, mobile process languages such as pi-ADL (Architectural Description Language) (Oquendo, 2004; Oquendo, 2005) – a derivative of high-order typed pi-calculus (Milner, 1999) is utilised to mitigate conceptual mapping and translation challenges between graph-based and block-structured models. The BPMSOA framework is instantiated for the role-based process modelling paradigm and evaluated (in particular for Role Activity Diagramming - RAD modelling (Ould, 1995)) with the objective of validating the hypothesis that mobile process languages, such as pi-ADL, can be utilised to enact role-based business process models in a GSOA environment. As a result, RADpiADL concrete program library is developed which enables the BPMSOA framework to semi-automatically generate code for a business process model in pi-ADL, which with minimal human intervention, can be executed in GSOA environment. Both the

BPMSOA framework and its role-based instantiation using the RADpiADL specifications provide novel contributions in bridging the above gap.

In the remainder of this paper we briefly present background and related work in the following section. In Section 3, an overview of the BPMSOA framework is presented followed by its role-based instantiation. In Section 4, we present the application of the role-based BPMSOA framework to the *Submission* process from the domain of digital libraries (DL) (Khan, et. al., 2006a). This application is evaluated using the static verification and dynamic validation described in Section 5, followed by critical analysis and reflection on the usefulness and limitations of the BPMSOA framework, role-based modelling approaches and pi-ADL in Section 6. Then in section 7 we draw conclusions and provide directions for future work.

2. BACKGROUND AND RELATED WORK

There are two aspects covered in this paper: i) mappings and translations between business process models and executable models, and ii) applying a homogeneous and structured approach to accomplish the (i) aspect. In relation to the first aspect, there have been efforts to map business process models, and in particular RADs (Ould, 1995), into system models using heuristics (Odeh, et. al, 2002; Odeh, et. al, 2003), simulations (Martinez-Garcia, et. al, 2002) and formal representations (Badica, et. al, 2001). However, such approaches require tools and technological support for enacting and evaluating the given business processes. For example, no tools have been identified to actually instantiate and execute business process models based on these approaches. Also, these approaches are not based on the SOA paradigm (i.e. web or grid services) which is considered to be a key shortcoming in dealing with business process management aspects. In this regard, many eXtensible Markup Language (XML) based business process modelling languages such as Business Process Modelling Language (BPML) (Arkin, 2002) and Business Process Execution Languages (BPEL) (Juric, et. al, 2004; Blanvalet, et. al, 2006) have been widely utilised to execute business processes using web services. However, the non-visual or non-diagrammatic nature of such languages (i.e. BPML, BPEL) makes them less suitable to understand, analyse and improve business processes by business analysts. In addition, these languages are designed for machine processing rather than for human understanding. This limitation has been overcome by mapping visual and graphical business process modelling language constructs to descriptive executable constructs, for example mappings between Fundamental Business Process Modelling Language (FBPML) and the Ontology Web Language for Services (OWL-S) (The OWLService Coalition, 2005; Guo, et. al, 2004), and translating Business Process Modelling Notations (BPMN) constructs into BPEL (White, 2005; Owen, et. al, 2007; White, 2007). These mappings and translations are not, however, straightforward and lack one-to-one construct mappings, which may become difficult to verify and validate. For example, researchers (Ouyang, et. al, 2006; Recker, et. al, 2006; BPMN, 2007) have identified limitations in translating graph-based BPMN models into block-structured BPEL and suggested various solutions. They argued that due to the conceptual difference between block-structured and graph-based models, it may not be possible to map all visual process topologies of BPMN and FBPML based process models to BPEL and OWL-S process descriptions, respectively (Guo, et. al, 2004; BPMN, 2007; Shapiro, 2002). RAD is another graph-based business process modelling language that we have attempted to map and translate into block-oriented pi-ADL using the BPMSOA framework, with the objective of assessing to what extent the channel unification and mobility characteristic of pi-ADL can facilitate mappings of graph-based models. It is important to mention here that we are not after comparing the strengths and weaknesses of different business process modelling languages and suggesting the suitability and appropriateness of a particular business process modelling language. Rather, we aim to experiment on selected case studies using the simplest approach such as RAD and provide an evidence of suitability of mobile process languages such as pi-ADL for enacting business process models in GSOA environments.

With respect to distributed computing and in particular grid SOA environment, mostly related work is in the domain of scientific workflow and there exist several tools and environments which facilitate execution of scientific workflows in grid SOA. For example, (Yu and Buyya 2006; Curcin and Ghanem 2008; Deelman et. al. 2009) have presented a comprehensive overview and taxonomy of workflow systems in distributed environments and in particular grid SOA such as Discovery Net, DAGMan, Pegasus, Traverna, Triana, Kepler, etc and languages such as UML and XML-based BPEL and SCUFL, etc. All these tools and respective languages support different programming models and can be used depending on the requirements of scientific applications.

In recent years, use of Model Driven approach for grid SOA has gained attention such as Grid Tool Kit (GDT) (Friese et. al. 2006) uses UML profiles and BPEL to model and execute scientific workflows. BPEL has gained good attention for workflow modeling not only by IT experts but also by domain experts, for example, BPEL extensions are introduced to adapt WSRF-compliant services for grid workflow modeling (Dornemann et. al. 2007). Despite its textual nature, mostly research specify workflows in graphical notations and many tools exist to translate graphical workflow representations into BPEL script such as XBaya (Shirasuna 2011), OMII-BPEL (OMII-BPEL 2011), Sedna (Wassermann et al., 2007), etc. However, according to some workflow domain experts e.g. (Deelman et. al. 2009), complex scientific workflows may require additional graphical rendering approaches such as nesting based on workflow hierarchies, concurrency, etc. This may be attributed to graph-oriented and block-structured conceptual difference between various modeling and workflow languages as discussed above. Among other approaches, (Blythe et. al. 2004) focused on artificial intelligence methods to automatically compose scientific workflows by applying semantic annotations. Similarly, (Brandic et. al. 2008) provide a holistic approach for QoS-aware workflows which enables scientific community to specify workflows using UML based graphical notations (and constraint annotations) in Tueta application (Pllana et. al. 2004), transform the workflow to XML-based language namely QoWL for workflow planning and execution in QoS-aware Grid Workflow Execution Engine to fulfill the user requirements. All stages of QoS-aware workflow are implemented and evaluated for Amadeus framework that provides a service-oriented environment but only supports acyclic workflow graphs. Manset et al (Manset et al. 2006) introduced gMDE architectural style that adopts Model Driven approach to generate programming models for IT experts to write and reuse modelled applications for grid environment. In contrast to the above approaches, A-WARE project (A-WARE 2008) enables domain/scientific experts to draw workflow using BPMN and transforms it into BPEL which is then handled by IT experts for service grounding and enactment of a scientific workflow in SOA grid environment (e.g. OGSA) and in particular UNICORE 6.

However, the above approaches are mostly intended for the IT experts but not for business domain experts or do not capture the full spectrum of various business process modelling approaches and languages. Further, despite some similarities there is a fine distinction between business process modelling and workflows. For example, (Dehnert and Van Der Aalst 2008) argued that workflows specifications are mostly intended for IT experts and may not be as intuitive as is a business process model for a business domain expert. Similarly, (Brandic et. al. 2007) distinguish scientific workflows for grid and business workflows based on the long execution time and large amount of data transfers in grid environment. This suggests that in order to maintain autonomy between business models and system models, a business process model requires mappings and transformation into workflow specifications for enactment in a GSOA environment. The BPMSOA framework handles the above limitations and requirement mostly from the business domain perspective and enables the transformation of a platform independent business process model into a concrete platform specific model using pi-ADL – an executable mobile process language - that can utilise grid enabled utility services. However, at the moment the lower-level grid based programming models, resource scheduling and monitoring are beyond the scope of the BPMSOA framework and handled as future research direction.

3. THE BPMSOA FRAMEWORK

This section briefly presents the BPMSOA framework (Khan, 2009). It consists of five logical layers as depicted in Figure 1. These are: (i) *Process Modelling,* (ii) *Generalisation,* (iii) *Transformation,* (iv) *Enactment* and (v) *Service Management and Integration*. Each layer (in particular the first four layers) takes a specific input and processes it using its internal computational components to generate a specific output which is then consumed by the subsequent lower layers. Further, each layer is provided with catalytic artefacts (i.e. Metamodels) to support the layered components in the intermediary translations of business processes. The role of the fifth layer (orthogonal to the other layers) is to manage and monitor the process of translating and enacting business processes.

The layered architecture of the BPMSOA framework is aligned with the separation of concerns design principle (Sommerville, 2006). These layers enable loose coupling between the activities of process modelling, generalisation, transformation and enactment for the objective of executing a business process model using application domain specific grid services. Below we briefly discuss each layer.

- The *Process Modelling* layer deals with process modelling view only. In addition, the *PM-Metamodel* is used to mask the syntactic heterogeneity of different business process modelling languages of a specific modelling approach by generalising the common modelling constructs, as briefly presented in (Khan et. al, 2008a). In this layer, a business process description can be modelled using a particular process modelling language. This process model can be used for comprehension, analysis and improvement purposes in addition to enactment.
- Similarly, the *Generalisation* layer translates a business process model into a machine-readable XML form whereas rules for mapping a process model into generic XML elements are defined in the associated instance of *α-Metamodel* (Khan et. al, 2008a) which is defined using a particular instance of *PM-Metamodel*. The output from the *Generalisation* layer can be used to exchange process information among different tools.
- The *Transformation layer* takes the above XML representation of the business process as an input to interpret and translate it into respective pi-ADL specification. The *τ-Metamodel* enables the *customisation* component to transform the XML representation into a programming context enabling the *concretisation* component to translate it into a respective pi-ADL form. The *χ-Metamodel* defines the pi-ADL specification for a particular business process modelling approach that is used by the *concretisation* component to translate the respective business process model into its corresponding pi-ADL code. Furthermore, the output of the *Transformation* layer is specific to a particular execution environment, and in particular pi-ADL environment e.g. emptyStore repository (ArchWare 2005).

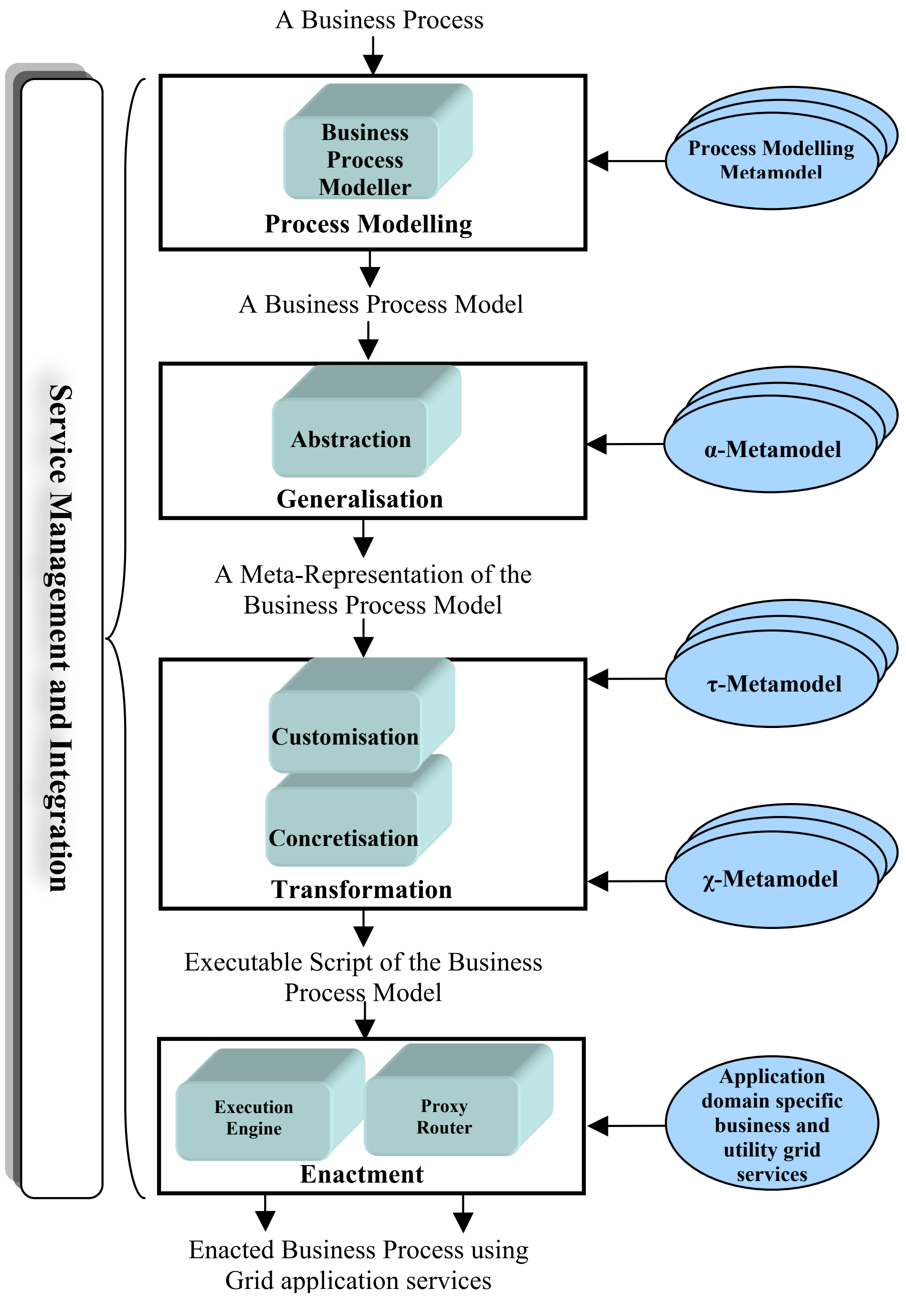
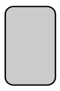
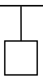
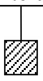
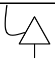
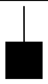
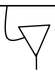
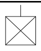
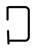

A Business Process
Business Process Modeller
Process Modelling
Process Modelling Metamodel
A Business Process Model
Abstraction
Generalisation
α-Metamodel
A Meta-Representation of the Business Process Model
Customisation
Concretisation
Transformation
τ-Metamodel
χ-Metamodel
Executable Script of the Business Process Model
Execution Engine
Proxy Router
Enactment
Application domain specific business and utility grid services
Enacted Business Process using Grid application services
Service Management and Integration

Figure 1: *The BPMSOA Architectural Framework*

- The *Enactment layer* takes the pi-ADL based executable script of the business process as an input and enacts it into an execution environment, in particular the ArchWare (ArchWare, 2005) virtual machine. The *proxy router* component enables the enacted process description to communicate with the external world. For example, it integrates the enacted process activities with respective application domain specific grid services using Java-based implementation of the proxy router (Khan, et. al., 2008b).
- The *Service Management and Integration* orthogonal architectural layer is supposed to bring higher levels of automation into the BPMSOA architectural framework i.e. facilitating the automated flow of information between the other four layers. Furthermore, this layer can also be utilised to keep track of the transformations of the business process at different levels of the BPMSOA framework and state transitions while in execution. However, the implementation of this layer is not within the scope of this paper and referred as future research direction.

Table 1: *Summary of the RAD Constructs Mappings at different Levels of BPMSOA*

RAD Notation	*PM-Metamodel* Meta-representation	*α-Metamodel*	*τ-Metamodel*	*χ-Metamodel*	pi-ADL Specifications and Abstractions
 Role	Composition Element (CE)	Role (SP, IP)	Composed Element (SP)	Intra-Role Mapping	*Role*
				Inter-Role Mappping	*IRegistry_Process Interaction_Repository Registry_Filler Router_Role_Connect*
 Part interaction	Receiver (FE), Interaction (IE)	Receiver_Part , Interaction, Role (SP, IP)	Interaction Element (SP)	Intra and Inter-Role Mappings	*Part_Interaction*
 Driver part interaction	Sender (FE), Interaction (IE)	Sender_Part, Interaction, Role (SP, IP)	Interaction Element (SP)		*Driver_Part_Interaction*
Trigger	Trigger – (Timer, Message, Error) (FE)	Trigger- (Timer, Message, Error) (SP)	Flow Element (SP)	Intra-Role Mapping	*RAD_Trigger, Trigger_External_Code*
State/Goal	State (FE)	State (SP)	Flow Element (SP)		*Abst_Goal_State, Abst_State_Action*
 Part Refinement	Gateway–Concurrency (FE)	Gateway - Concurrent (SP, FP)	Flow Element (SP)		*Part_Refinement, Part_Synchronisation, Part_Split, Part_Thread*
 Activity	Activity – Action (FE)	Activity - Action (SP)	Flow Element (SP)		*RAD_Activity, Activity_External_Code Proxy_Router (for grid services)*
 Case Refinement	Gateway-Conditional (FE)	Gateway - Conditional (SP, FP)	Flow Element (SP)		*Case_Activity, Case_Refinement, Case_Synchronisation, XOR_Split, Case_Thread*
 Role Instantiation	Activity – Instantiation (FE)	Activity - Role-Instantiation (SP)	Flow Element (SP)		*Instantiate_Role*
State	State (FE)	State (SP)	Flow Element (SP)		Using *in_trig* and *out_trig* channels by applying unifications
 Loop	Gateway-Iterative (FE)	Gateway-Iterative (SP, FP)	Flow Element (SP)		*Iterative_flow, normal_flow, Loop_handling_with_Activity (inside Case_Thread)*
Process	CE, IE, FE	SP, IP, FP	SP, NFP, CP, AIP	Inter-Process	*Enactment Engine*
				Intra-Process	*Process Controller*

Hence, BPMSOA's layered architecture follows a general to specific model translation of business processes, which suggests that BPMSOA is closely aligned with the MDE approach i.e. that of transforming platform independent business process models into concrete platform dependent models. Moreover, the PM-, *α-*, τ- and χ-*Metamodels* (as depicted in Figure 1) play two important roles according to the needs of metamodels in MDE (Stuart, 2002). Firstly, they provide syntactic and semantic guidelines to translate a given business process model at respective layers of the BPMSOA framework. Without these guidelines, the BPMSOA components cannot dynamically translate different process designs. Each metamodel is associated with a specific component of a particular layer, for example the *α-Metamodel* provides translation rules and guidelines for the *abstraction* component. Similarly, the *τ-Metamodel* and the χ-*Metamodel* are associated with the *customisation* and *concretisation* components respectively. Secondly, there can be multiple implementations of associated metamodels at the respective layers, for example one implementation of *PM-Metamodel* can be used to capture the role-based perspective whereas another one can consider the functional view of business processes.

However, a particular implementation of the *PM-Metamodel* requires a corresponding implementation of the *α-Metamodel,* the *τ-Metamodel* and the χ-*Metamodel.* Whilst these layers with the multiple implementations of metamodels enable the BPMSOA framework to be general, extensible and flexible, this framework can become rigid and thus requires proper information flow between layers in addition to coordination. In this regard, the orthogonal *Service Integration and Management* layer is introduced which can be used to coordinate the information flow at the appropriate levels and verify the correctness of respective translations.

In this paper, the role-based business process modelling approach (and in particular Role Activity Diagramming (Ould, 1995) is adopted, which provides a distinctive modelling approach because of its intuitiveness, and its increased visibility by depicting flexible process topologies and event-driven and collaborative perspectives (Harrison-Broninski, 2007). Hence the role-based BPMSOA is instantiated and role-based metamodels and components of BPMSOA framework have been published elsewhere and can be accessed from (Khan, 2009; Khan, et. al, 2007; Khan, et. al, 2008a; Khan, et. al, 2008b; Khan, et. al, 2008c). These metamodels capture different perspectives built using different elements as depicted in Annex-I. All perspectives (i.e. Structural, Interaction, Flow, Control, Non-Functional and Application Integration Perspectives (SP, IP, FP, CP, NFP, AIP respectively)) and their elements (i.e. Composition, Flow and Interaction Elements (CE, FE, IE respectively)) have been presented in detail in (Khan, 2009). Due to space limitations, in Table 1 we summarise just the mappings of the different RAD constructs while being processed at the different role-based BPMSOA layers and facilitated by the respective layer's metamodels.

4. THE CASE-STUDY: APPLYING THE ROLE-BASED BPMSOA FRAMEWORK ON THE DL SUBMISSION PROCESS

In order to aid understanding of the functionality of BPMSOA, an enactment process (as detailed below) is applied on the *Submission* process from the digital libraries (DL) domain. The *Submission* process is introduced to enact more complex RAD process models with control flow structures such as loops, case and part refinements, and hanging threads, etc. The *Submission* process is concerned with submitting an article or manuscript for publishing in a particular digital library. The manuscript is submitted to the *Editor* (role) by the *Author* (role). The *Editor,* after having performed the initial evaluation, notifies the *Author*, if the manuscript is not accepted, to proceed to peer-review. The *Author* subsequently then may consider resubmitting the same article after a further review of the initial feedback received. The articles which are accepted for peer-review are properly managed (i.e. assigned an ID, etc.) and forwarded for peer-review process. Finally, the *Author* is notified of progress while the article is being reviewed. The following is a step-by-step progression of above mentioned enactment process when applied on the *Submission* process.

- Step 1: Modelling the Submission Process: Using the above description, the RAD model of the Submission process that captures the basic flow of activities is depicted in Figure 2(a). RADModeller Ver. 1.1.0 (RADModeller, 2006) is used as the process modeller component of the Process Modelling layer. There are two roles involved in this process: Author and Editor. Activities such as "Peer-review Article" in the Editor role is shown as an encapsulated activity, as it may involve other roles such as a Reviewer role that does not fall within the region of the Submission process.
- Step 2: Generating the generalised XML representation of the Submission Process: Using BPMSOA's role-based α-Metamodel and abstraction component, the XML representation of the Submission process is generated and a snippet is represented in Figure 2(b). Once again, RADModeller Ver 1.1.0 is used as the abstraction component of the Generalisation layer.
- Step 3: Translating the process model into pi-ADL specifications: Using the role-based τ-Metamodel and customisation component, the XML representation of the Submission process is translated to the Java programming language, which can facilitate the concretisation component in translating it into pi-ADL. Using the role-based χ-Metamodel, the RAD pi-ADL specifications have been generated (Khan 2009) which can be utilised to program role-based business process models into pi-ADL. Due to space limitations these RAD pi-ADL specifications cannot be presented here but a small example is depicted in Annex-II. Using these RAD pi-ADL specifications, the Submission process is translated into pi-ADL (over 2000 lines of code)[*] and accessible from (Khan 2009). The signatures of more specific pi-ADL abstractions from the RAD pi-ADL specifications, which have been used to translate specific RAD constructs of the Submission process model, have been presented in Table 2. In Table 2, the application of the RAD-based pi-ADL specifications for the different modelling constructs can be classified into the following three categories:
    - *Fully-automated:* reflects that a RAD modelling construct can be automatically translated into the corresponding pi-ADL abstractions. For example, *Proxy_Router, IRegistry_Process, Interaction_Registry, Router_Role_Connect, Process_Engine_Enactment, Driver_Part_Interaction, Part_Interaction, Part_Registry*, and utility functions, etc;
    - *Semi-automated:* shows that a RAD modelling construct requires human intervention to add specific elements to the corresponding RAD pi-ADL abstractions, such as *Role, Activity_External_Code, Case_Activity, RAD_Trigger*, etc.

[*] Due to space limitations, the pi-ADL code for the Submission process is either accessible from (Khan 2009) or downloadable from http://www.cems.uwe.ac.uk/~zakhan/BPMSOA/processes.pdf.

For example, defining parameters and their values for a specific instance of *Activity_External_Code* of a process activity which are required to invoke a specific grid service. In addition, *Case_Activity, RAD_Trigger, Instantiate_Activities* (sub-abstraction or Role for handshaking protocols), etc. need to declare global channels which can be utilised for human user interaction for specific roles. Similarly, in order to handle loops, *Loop_handling_with_Activity* and *iterative_flow* abstractions are defined which utilise global channels to demonstrate iterative behaviour; and,

- *Customised:* represents a RAD modelling construct which requires humans (i.e. pi-ADL programmers) to define the required behaviour of that modelling construct in the respective pi-ADL specification. For example, a driver part interaction may receive a resource value from a human user and may compose it into *Resource_Abst* which would require behavioural changes in the default *Driver_Part_Interaction* abstraction.

- Step 4: Executing the pi-ADL specification of the Submission Process: When the pi-ADL specification of a business process model is enacted using the ArchWare environment (ArchWare, 2005), the direct and indirect outputs are obtained. The direct output (i.e. human-based activities) of the execution of the Author and the Editor roles of the Submission process is depicted in Figure 3. In Figure 3, human users enter input values for specific activities within quotes (i.e. " "), whereas the system generated output is depicted without any quotes. Also, both the Author and the Editor roles are executed in the required order as labelled with the sequence numbers and associated descriptions. The handshaking protocol is used to ensure that the human user is connected with a user interface to interact with the required process activities at run-time. Furthermore, it is also depicted in Figure 3 that both of these roles execute loop, case refinements and nested part refinement constructs. The execution output also reveals that the pi-ADL based process activities successfully communicate with the ASP component of the proxy router (Khan, 2009), which enables them to be integrated with the respective grid-aware services, for example being deployed using the Globus Toolkit 4 (Sotomayor, 2005). In contrast, the indirect outputs (i.e. the detailed execution and communication logs of the Submission process and the ASP component of the proxy router) are used for the evaluation of the dynamic behaviour of the Submission process model as discussed in the following section.

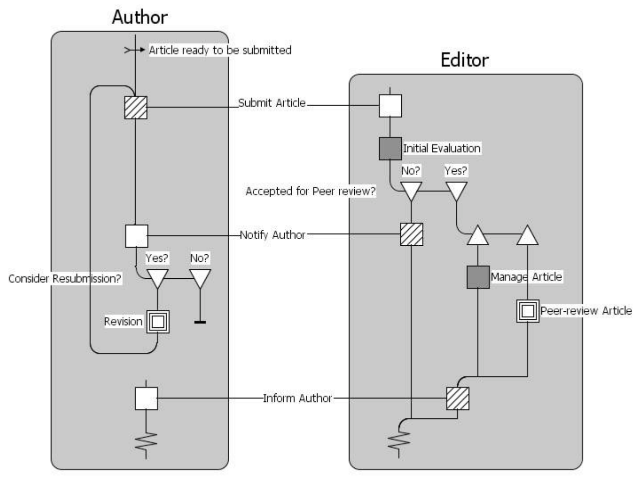
5. EVALUATION AND DISCUSSION

In order to evaluate the BPMSOA framework, a structured evaluation methodology has been derived from the research hypothesis (section 1) (Khan, 2009) and is briefly presented below. In general, this evaluation is mainly concerned with the static verification of the correct syntax of the translations which take place at the different layers of the BPMSOA framework. In addition, it also performs the dynamic validation of the correct behaviour of the given business process when enacted. The overall goal of the combined static verification and dynamic validation is to assess and reflect on the extent to which the BPMSOA framework can be utilised to bridge the gap between business process models and GSOA environments. Both static verification and dynamic validations are based on the Kotonya and Sommerville's concern-based approach (Kotonya, et. al, 2002) to derive main concerns which are verified and validated using the checklist questionnaires and walkthrough techniques (Sommerville, 2006).

5.1. Static Verification

Static verification is performed in two stages. In the *first stage*, the translation from a business process model to the respective abstract generalised form (i.e., RAD to XML) is verified with respect to the role-based *α-Metamodel*. In the *second stage*, the translation from the generalised process form to the concrete and executable form (i.e., XML to pi-ADL) is verified with respect to the pi-ADL specifications of RADs. In general, the above two verification stages conform to the expression 'Are we building the product right?' (Sommerville, 2006) i.e. do the respective translations conform to the guidelines of the respective metamodels at the different layers of BPMSOA? Using the concern-based approach (Kotonya, et. al, 2002), Figure 4 depicts the main concerns related to static verification. In Figure 4, verification stage I requires the quantitative analysis of RAD business process models, with the objective of cross-verifying that the modelling in a given RAD process model tallies with its respective XML modelling elements. Figure 4 also derives the main requirements and associated questionnaires for verifying the correctness of the resultant pi-ADL specifications of the translated RAD XML constructs. Based on these concerns different RAD pi-ADL abstractions, newly defined data types and channel unifications are verified using walkthrough/inspection technique (Sommerville, 2006).

As a result of the derived questions (as depicted in Figure 4), a checklist for necessary mappings has been prepared using different RAD pi-ADL abstractions, newly defined data types (a glimpse is depicted in Table 3 and presented in detail in (Khan, 2009)) and channel unifications, which are verified using walkthrough/inspection technique (Sommerville, 2006). A few checklist questionnaires to assist this verification have been shown in Table 4.



a) RAD model of *Submission* process

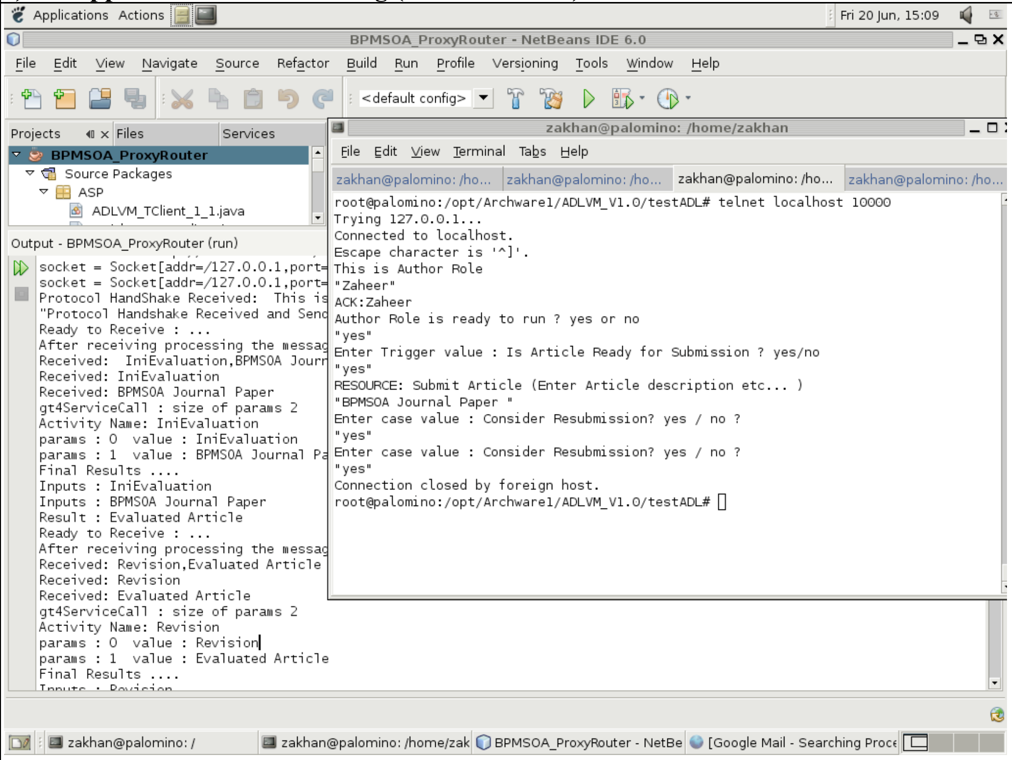
```
<CE type="Role", id="1", desc="Author">
  <FE  Id="2", type ="Trigger", desc="Article ready to be submitted",
        NextElement = "3" />
  <FE  Id="3", type ="Sender-Part", Interaction_Id="30", NextElement="4",
        PreviousElement = "2_10", Gateway_Id="10"/>
  <FE Id="4", type = "Receiver-Part", Interaction_Id="31",
        NextElement ="5", PreviousElement="3" />
                          ::::::

<CE type="Role", id="14", desc="Editor">
  <FE  Id="15", type="Reciever-Part", Interaction_Id="30", NextElement="16" />
                          :::::::
<IE id="30", type="Interaction", desc="Submit Article">
   <FE Id="3", type="Sender-Part", Parent_CE_Id="1" />
   <FE Id="15", type="Receiver-Part", Parent_CE_Id="14" />
</IE>
                          :::::::
```

b) XML snippet of the *Submission* process

```
:::::
"via debugConn receive { Role : Author  }"
"via debugConn send { Role: Author }"
                  :::::
"via prox_ch send { Revision, Evaluated Article }"
                  :::::
"via com_ch send { Enter case value :Accepted for Peer-review? yes / no ? }"
"via com_ch receive { no }"
"via Sin_trig receive { no, Evaluated Article }"
                  :::::
"via channel_value1 receive { true, Article is managed and Indexed }"
"via out_trig send { true, Article is managed and Indexed }"
                  :::::
```

c) A snippet of the Trace-File Log (over 2000 lines)



Applications Actions
Fri 20 Jun, 15:09
BPMSOA_ProxyRouter - NetBeans IDE 6.0
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
<default config>
Projects
Files
Services
BPMSOA_ProxyRouter
Source Packages
ASP
ADLVM_TClient_1_1.java
Output - BPMSOA_ProxyRouter (run)
zakhan@palomino: /home/zakhan
File Edit View Terminal Tabs Help
zakhan@palomino: /ho...
zakhan@palomino: /ho...
zakhan@palomino: /ho...
zakhan@palomino: /ho...
zakhan@palomino: /
zakhan@palomino: /home/zak
BPMSOA_ProxyRouter - NetBe
[Google Mail - Searching Proce

```
socket = Socket[addr=/127.0.0.1,port=
socket = Socket[addr=/127.0.0.1,port=
Protocol HandShake Received:  This is
"Protocol Handshake Received and Send
Ready to Receive : ...
After receiving processing the messag
Received:  IniEvaluation,BPMSOA Journ
Received: IniEvaluation
Received: BPMSOA Journal Paper
gt4ServiceCall : size of params 2
Activity Name: IniEvaluation
params : 0  value : IniEvaluation
params : 1  value : BPMSOA Journal Pa
Final Results ....
Inputs : IniEvaluation
Inputs : BPMSOA Journal Paper
Result : Evaluated Article
Ready to Receive : ...
After receiving processing the messag
Received: Revision,Evaluated Article
Received: Revision
Received: Evaluated Article
gt4ServiceCall : size of params 2
Activity Name: Revision
params : 0  value : Revision
params : 1  value : Evaluated Article
Final Results ....
```

```
root@palomino:/opt/Archware1/ADLVM_V1.0/testADL# telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
This is Author Role
"Zaheer"
ACK:Zaheer
Author Role is ready to run ? yes or no
"yes"
Enter Trigger value : Is Article Ready for Submission ? yes/no
"yes"
RESOURCE: Submit Article (Enter Article description etc... )
"BPMSOA Journal Paper "
Enter case value : Consider Resubmission? yes / no ?
"yes"
Enter case value : Consider Resubmission? yes / no ?
"yes"
Connection closed by foreign host.
root@palomino:/opt/Archware1/ADLVM_V1.0/testADL# 
```

d) A snippet of the Proxy Router – ASP Log with Author Role

Figure 2: *Translations and execution logs of the Submission process using the BPMSOA*

Table 2: *pi-ADL Abstractions of the Submission Process*

RAD Modelling Construct		Corresponding RAD pi-ADL Abstraction(s) (actual pi-ADL code is over 2000 Lines of Code)	Classification of RAD pi-ADL Abstractions
Concerned Elements	**XML Model Elements**		
Modelling constructs of the Author Role	Trigger (Article Ready)	RAD_Trigger using normal_flow, iterative_flow and abst_comp	Semi-automated
	Sender-Part (Submit Article)	Driver_Part_Interaction	Customised
	Receiver-Part (Notify Author)	Part_Interaction	Fully-automated
	Conditional-Activity (Consider Resubmission?)	Case_Activity	Semi-automated
	Gateway-Condition, Case-Thread (Yes?), Case-Thread (No?)	Case_Refinement, XOR_Split, Case_Synchornisation, Case_Thread1, Case_Thread2	Customised and Semi-automated
	Activity-Encapsulation (Revision)	RAD_Activity, Activity_Internal, and Activity_External_Code	Semi-automated
	Gateway-Iterative	Normal_flow, iterative_flow, and abst_comp sub-abstractions	Semi-automated
	Activity-Stop (Stop)	Stop abstraction	Fully-automated
	Receiver-Part (Inform Author)	Part_Interaction1	Customised and Semi-automated
Author Role		AuthorRole	Semi-automated
Modelling constructs of the Editor Role	Receiver-Part (Submit Article)	Part_Interaction	Fully-automated
	Activity-Action (Initial Evaluation)	RAD_Activity, Activity_Internal, and Activity_External_Code	Semi-automated
	Conditional-Activity (Accepted for Peer-review?)	Case_Activity1	Semi-automated
	Gateway-Condition, Case-Thread (No?), Case-Thread (Yes?)	Case_Refinement1, XOR_Split, Case_Synchornisation, Case_Thread3, Case_Thread4	Semi-automated
	Sender-Part (Notify Author)	Driver_Part_Interaction1	Fully-automated
	Gateway-Concurrent, Part-Thread1, Part-Thread2.	Part_Refinement, Part_Split, Part_Synchronisation, Part_Thread1, Part_Thread2.	Semi-automated
	Activity-Action (Manage Article)	RAD_Activity, Activity_Internal, and Activity_External_Code	Semi-automated
	Activity-Encapsulation (Peer-review Article)	RAD_Activity, Activity_Internal, and Activity_External_Code	Semi-automated
	Sender-Part (Inform Author)	Driver_Part_Interaction1	Fully-automated
Editor Role		EditorRole	Semi-automated
Submission Process		Proces_Engine_Enactment, Process_Controller, Router_Role_Connect, IRegistry_Process, Proxy_Router, Utility Functions, Customised Data Types	Semi-automated

5.2. Dynamic Validation

Dynamic validation is used to evaluate the behavioural correctness of the respective pi-ADL specifications of the RAD business processes at run-time which generates *direct* and *indirect* outputs. The aim here is to reveal the presence of any behavioural errors and limitations in the respective pi-ADL specification in order to suggest improvements accordingly. In addition, this validation is used in assessing the suitability of pi-ADL as the mobile process execution language adopted in the BPMSOA framework. This behavioural validation conforms to the expression 'Are we building the right product?' (Sommerville, 2006) i.e. is the process model specification that an end user or a client expects correctly propagated to the pi-ADL level? Using the concern-based approach (Kotonya, et. al, 2002), Figure 5 derives the main concerns and associated questions to validate the behavioural correctness of the resultant pi-ADL specifications by enacting the *Submission* process model. Based on these concerns, checklist questions have been derived (a few question are depicted in Table 5) and utilised in dynamically validating the behaviour of RAD process models.

The pi-ADL specification of the *Submission* process is enacted and executed using the *execution engine* component and an execution log (i.e. *indirect* output) is stored in a trace-file (only a snippet is shown in Figure 2(c) as the actual log is over 2000 lines and accessible from Khan 2009). This execution log is made more expressive by using a debugger channel (i.e. debugConn) that records the communication between the different channels of the RAD pi-ADL abstraction instances. Furthermore, the ASP component of the *proxy router* stores the log of the communication between the RAD process activities and the respective services (a snapshot of ASP log is shown in Figure 2(d)). Both of the above logs serve as *indirect* outputs for the process being executed. In addition, the output generated to have human-user interaction during the execution of the *Submission* process model is used as a *direct* output and has been presented in Figure 3.

Table 3: *pi-ADL Abstractions and Data Types for some of the Role, Hanging Thread and Process Modelling Constructs of RAD Modelling (Khan, 2009; Khan, et. al, 2008c)*

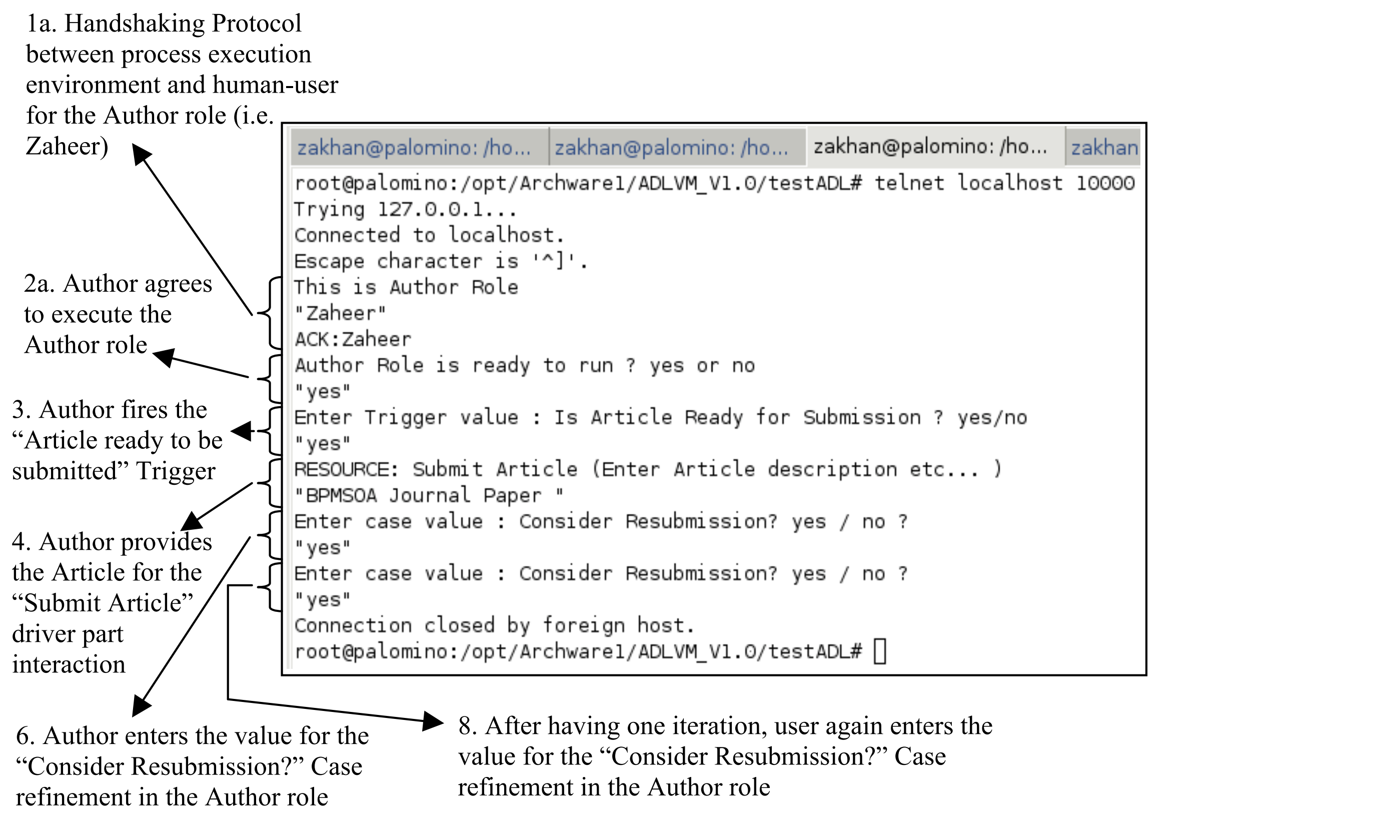
Concerned Elements (RAD Constructs)	**New pi-ADL Constructs (or Abstraction)**	**Additional supporting pi-ADL Abstractions**	**New Data Types**
Role	*Role, router_role_connect*	*Iregistry_Process Interaction_Repository Registry_Filler*	*IR_Tuple, Trans_Element, Role_Address, globally scoped Role Channel(s)*
Hanging Thread	-	*Customised FE abstractions, for_hangthread n*	*Role scoped channels*
Process	*Process_Engine_Enactm ent*	*Process_Controller, IRegistry_Process, Proxy_Router*	*Globally declared channels*

Table 4: *A Few Static Verification Checklist Questions for Role Modelling Construct of RAD (Khan, 2009; Khan, et. al, 2008c)*

Modelling constructs and additional pi-ADL abstractions	**Concerned Checklist Questions**
Role	1. Does each role abstraction consist of abstractions of individual RAD constructs? 2. Have RAD constructs inside the role abstraction been enacted? 3. Have the channels of RAD constructs inside a role been unified as per the order depicted in the RAD process model? 4. Have the mechanisms in a role abstraction been defined to handle more than one hanging threads? 5. Has a global role_channel been declared for each role to communicate with human actors? E.g. trigger value, Case_Activity value etc. 6. Has a protocol been defined to start role processing? 7. Has a Part_Registry been enacted within a role abstraction to handle part interactions? 8. Have all roles registered their names and channels to Interaction Registry at bootstrapping so that they can receive transmission element?

Table 5: *Dynamic Validation Checklist Questions For a few RAD Modelling Constructs (Khan, 2009; Khan, et. al, 2008c)*

Modelling Constructs	**Concerned Questions for the Expected Behaviour**
Role	1. Has the role performed the required behaviour during execution? 2. Has the topological formation within role correctly executed? i.e. flow of activities. 3. Have all the hanging threads executed properly?
Activity and Encapsulation	4. Has the activity performed the required behaviour during execution? 5. Has the activity properly linked with its corresponding grid service?
Trigger	6. Has the trigger performed the required behaviour during execution?
Stop	7. Has stop activity provided the required behaviour during execution?
Driver-part interaction	8. Has the driver part interaction transmitted the resource and transmission element to its corresponding receiver-part interaction in another role?
Receiver-part interaction	9. Has the receiver part interaction received the resource being sent by its corresponding driver-part interaction from another role?



1a. Handshaking Protocol between process execution environment and human-user for the Author role (i.e. Zaheer)

2a. Author agrees to execute the Author role

3. Author fires the “Article ready to be submitted” Trigger

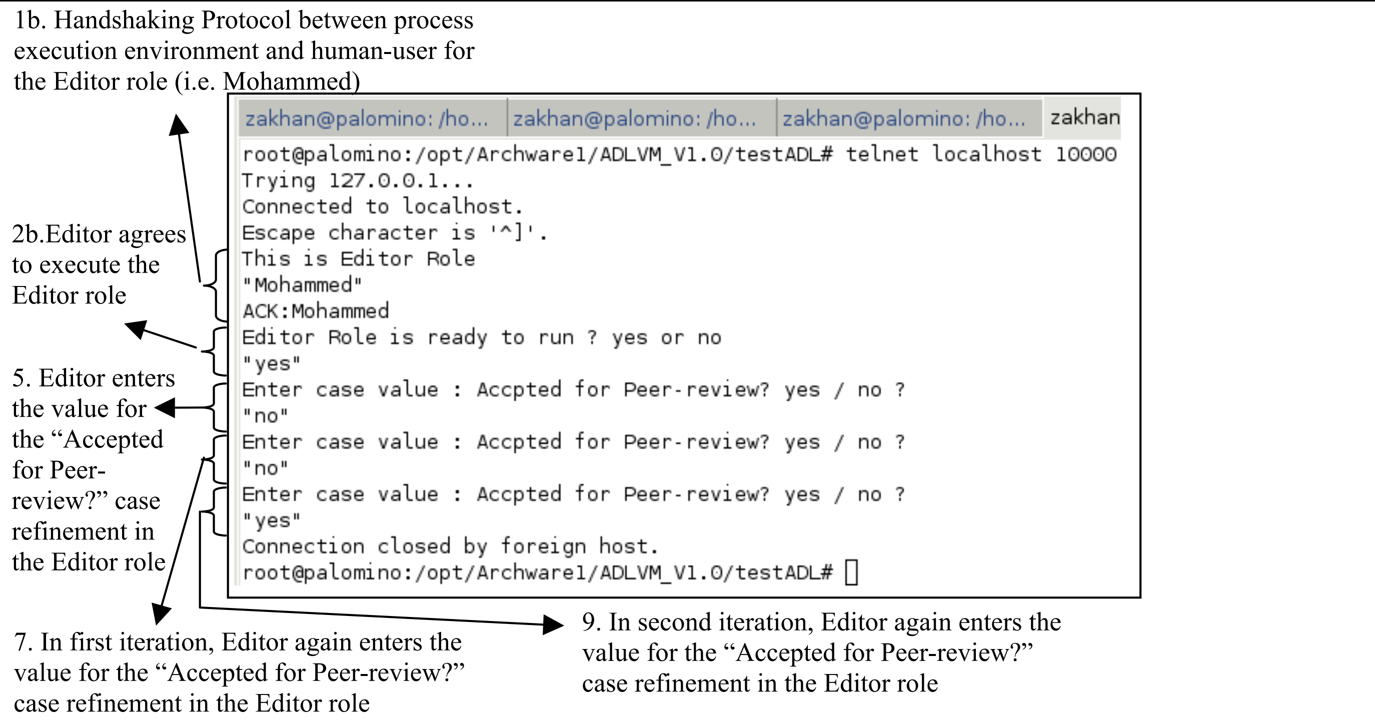
4. Author provides the Article for the “Submit Article” driver part interaction

6. Author enters the value for the “Consider Resubmission?” Case refinement in the Author role

8. After having one iteration, user again enters the value for the “Consider Resubmission?” Case refinement in the Author role

```
root@palomino:/opt/Archware1/ADLVM_V1.0/testADL# telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
This is Author Role
"Zaheer"
ACK:Zaheer
Author Role is ready to run ? yes or no
"yes"
Enter Trigger value : Is Article Ready for Submission ? yes/no
"yes"
RESOURCE: Submit Article (Enter Article description etc... )
"BPMSOA Journal Paper "
Enter case value : Consider Resubmission? yes / no ?
"yes"
Enter case value : Consider Resubmission? yes / no ?
"yes"
Connection closed by foreign host.
root@palomino:/opt/Archware1/ADLVM_V1.0/testADL# 
```

a) Output of the Author Terminal



1b. Handshaking Protocol between process execution environment and human-user for the Editor role (i.e. Mohammed)

2b.Editor agrees to execute the Editor role

5. Editor enters the value for the “Accepted for Peer-review?” case refinement in the Editor role

7. In first iteration, Editor again enters the value for the “Accepted for Peer-review?” case refinement in the Editor role

9. In second iteration, Editor again enters the value for the “Accepted for Peer-review?” case refinement in the Editor role

```
root@palomino:/opt/Archware1/ADLVM_V1.0/testADL# telnet localhost 10000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
This is Editor Role
"Mohammed"
ACK:Mohammed
Editor Role is ready to run ? yes or no
"yes"
Enter case value : Accpted for Peer-review? yes / no ?
"no"
Enter case value : Accpted for Peer-review? yes / no ?
"no"
Enter case value : Accpted for Peer-review? yes / no ?
"yes"
Connection closed by foreign host.
root@palomino:/opt/Archware1/ADLVM_V1.0/testADL# 
```

a) Output of the Editor Terminal

Figure 3: *Execution Output of the Submission Process*

5.3. Evaluation Outcomes

Both static verification and dynamic validation have identified a number of issues related to RAD process model translation and enactment using the BPMSOA framework, for example:

- Stage I verification verifies that the RAD model of the *Submission* process is correctly translated into its corresponding XML representation since the number of elements, their description and order in the XML representation is consistent with the corresponding RAD model (Khan, 2009). Also, inspecting the respective pi-ADL specifications of a RAD process model based on the stage II criteria verifies that the required functionality is correctly specified in the pi-ADL abstractions of the specific modelling constructs. However, stage II verification reveals that translating a RAD modelling construct into the respective pi-ADL specification *is not straightforward* since it requires customisation during translations for specific modelling constructs, for example, parametric values for RAD activities, resource elements (i.e. messages, e.g. 'Submit Article' driver part interaction of the *Submission* process), variation of control structures (i.e. loops, case and part refinements), human-user based activities, etc. (Khan, 2009). Such translation customisations lessen the extent to which RAD process models can be automatically translated into the respective pi-ADL specifications.

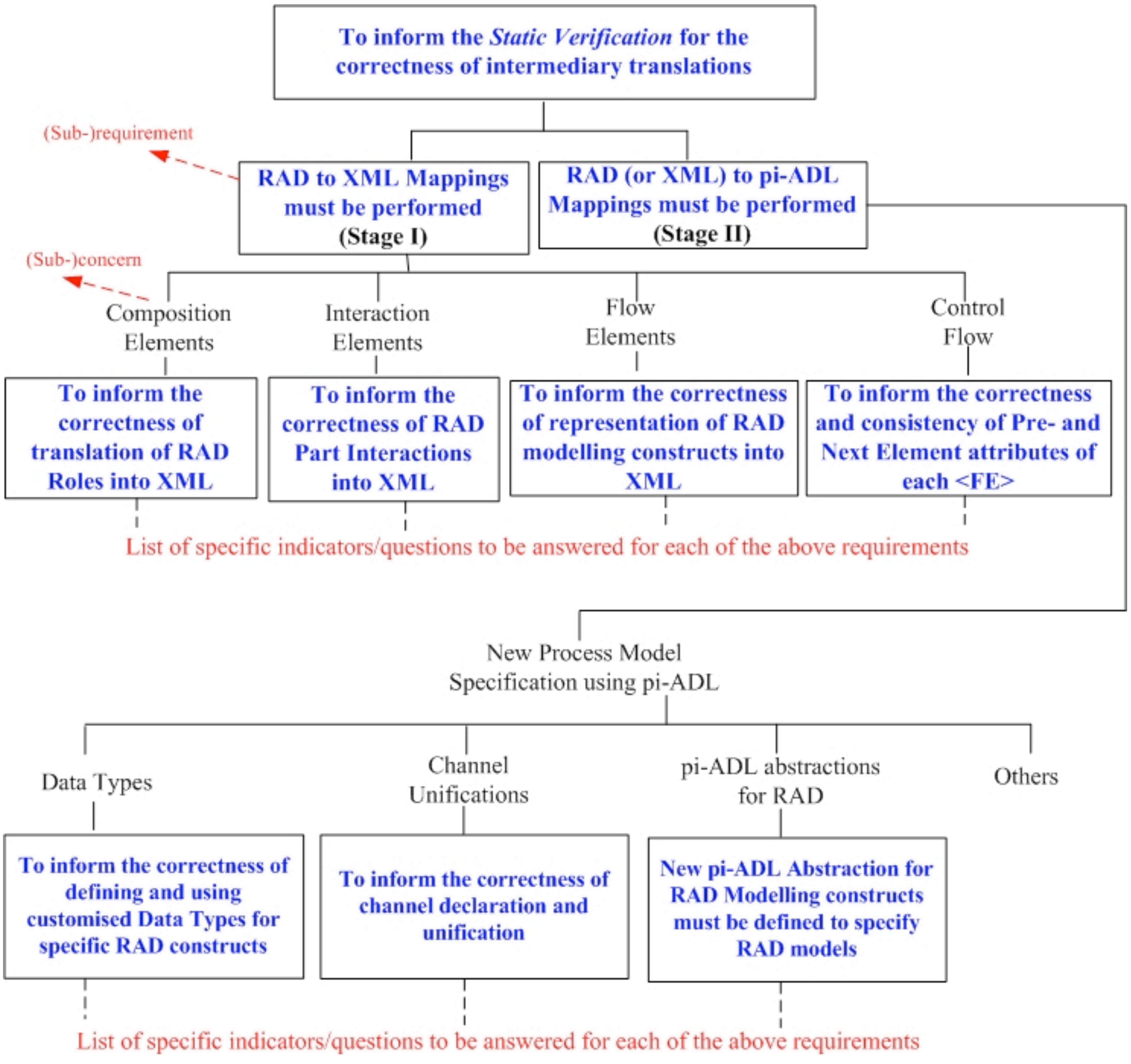

To inform the Static Verification for the correctness of intermediary translations
(Sub-)requirement
RAD to XML Mappings must be performed (Stage I)
RAD (or XML) to pi-ADL Mappings must be performed (Stage II)
(Sub-)concern
Composition Elements
Interaction Elements
Flow Elements
Control Flow
To inform the correctness of translation of RAD Roles into XML
To inform the correctness of RAD Part Interactions into XML
To inform the correctness of representation of RAD modelling constructs into XML
To inform the correctness and consistency of Pre- and Next Element attributes of each <FE>
List of specific indicators/questions to be answered for each of the above requirements
New Process Model Specification using pi-ADL
Data Types
Channel Unifications
pi-ADL abstractions for RAD
Others
To inform the correctness of defining and using customised Data Types for specific RAD constructs
To inform the correctness of channel declaration and unification
New pi-ADL Abstraction for RAD Modelling constructs must be defined to specify RAD models
List of specific indicators/questions to be answered for each of the above requirements

Figure 4: *Static Verification using the Concern-based Approach*

- In addition, the static verification (using inspections) of the pi-ADL specifications of the *Submission* process reveals that all the channel unifications have been correctly performed. An invalid channel unification between different abstractions may result in an unexpected process behaviour (e.g. halting the process). Such an error may be caused due to improper design of process abstractions or an incorrect channel qualifier name checking (i.e. syntactic checking), which cannot be detected at compile time. This indicates that channel unifications must be verified for process abstractions such as process activity, proxy router, case-refinements, loops, etc.

In general, the static verification of the role-based BPMSOA instantiation verifies that the translation of the *Submission* process from RAD process models to the respective pi-ADL specifications (through the intermediate XML translation steps) is syntactically correct with respect to the checklist questions derived using the adopted concern based approach. Stage II verification has revealed a number of issues which restrict the extent to which an XML representation of a RAD process model can be automatically translated into the respective pi-ADL specification. These issues can be mainly attributed to coarse grained RAD process models with poor execution semantics, and the restrictions in designing specific modelling constructs such as nested gateways (Khan, 2009).

Similarly, based on the above dynamic validation for the *Submission* process (Khan, 2009), the following few observations and results can be briefly reported and reflected on:

- Pi-ADL has the capability to capture and model different execution semantics, but to automate the correct interpretation of a coarse-grain RAD process model is not a straightforward procedure. The dynamic validation of the *Submission* process reveals that pi-ADL can model the structural, informational, flow and behavioural aspects of a business process (Khan, 2009). However, it has been observed that the more enriched RAD business process models are the more they can lead to correct and clear execution semantics. The following few scenarios highlight the uncertainty in determining the required execution behaviour for a particular modelling construct due to the unavailability of certain information in the corresponding RAD process models which may result in unwanted behaviour.
  - It is difficult to automatically determine at run-time whether the resource element of a driver-part interaction needs to be entered by a human-user of that role or that resource received from a previous activity or modelling construct. This reveals that there should be an application domain specific data model associated with the business process models that can be utilised during the process model execution. However, current models' representations (i.e. XML and pi-ADL specification of RADs) at different levels of BPMSOA cannot automatically generate such resources.
  - Similarly, a goal-state and/or merging two states does not provide clear execution semantics that can be automatically determined from the respective RAD model. For example, in the part refinement segment (in the *Editor* role of the *Submission* process model), all threads produce some state values which may need to be passed to an activity after the part refinement segment. However, the RAD model of the *Submission* process does not indicate how to synthesize these multiple state values generated by the part refinement threads before these are passed on to the next activity.
- The current RAD pi-ADL specifications necessitate process activities of the enacted process model to assemble the parameters required to invoke a particular grid service. A run-time error in integrating a process activity with the respective grid service (for example, due to incorrect parameters) can result in erroneous process execution behaviour such as halting the process model execution (Khan, 2009). For example, based on the *intra-role* model of *χ-Metamodel* (Khan, et. al, 2007) when a process activity goes into blocking mode (i.e. the synchronous *receive* prefix is waiting for a reply from the *Proxy_Router* abstraction), it also keeps the execution of the subsequent process modelling constructs blocking since they cannot continue their execution because they are waiting for the *control* trigger and *state* values from the current process activity. The current RAD pi-ADL specifications do not deal with such a blocking behaviour due to the default synchronous behaviour of pi-ADL *send/receive* prefixes. This suggests that further investigation is required in re-designing pi-ADL abstractions and handling such run-time errors.

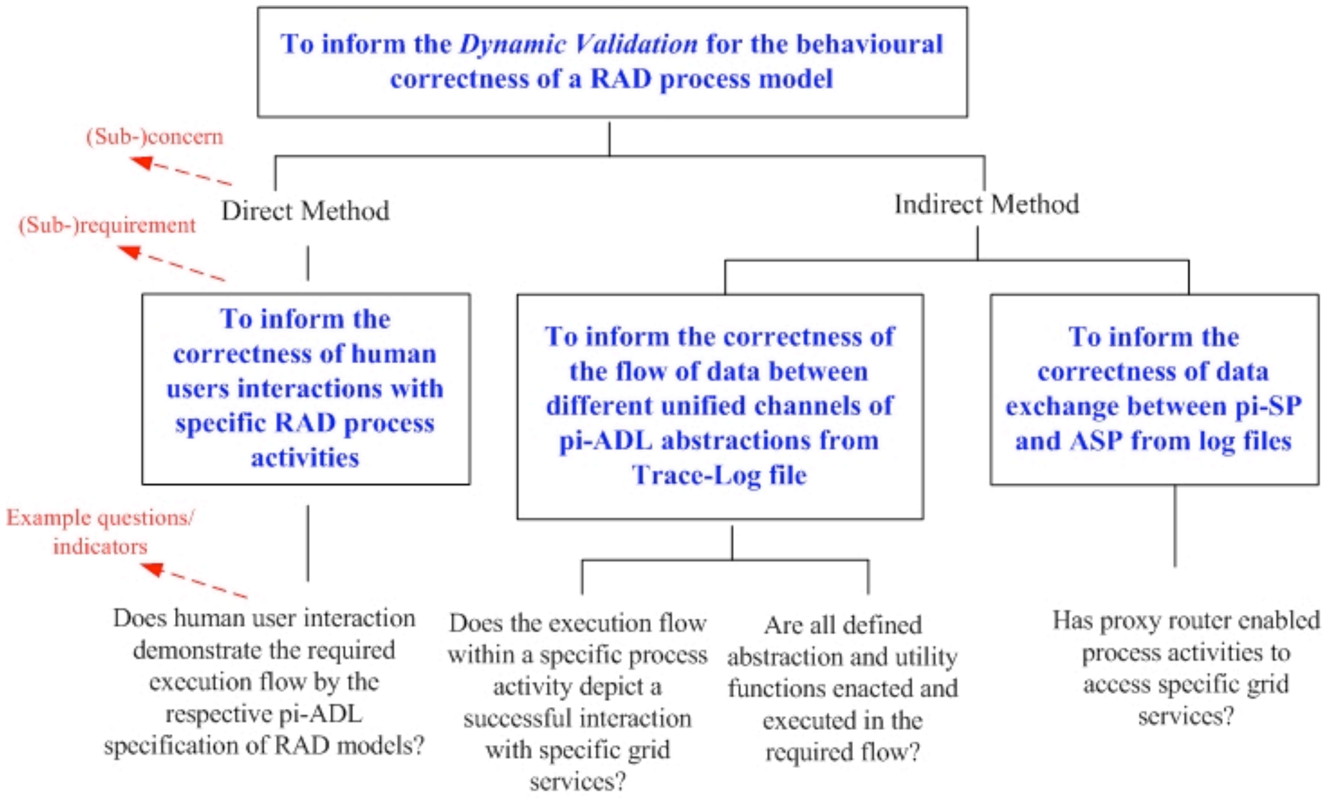

To inform the *Dynamic Validation* for the behavioural correctness of a RAD process model
(Sub-)concern
Direct Method
Indirect Method
(Sub-)requirement
To inform the correctness of human users interactions with specific RAD process activities
To inform the correctness of the flow of data between different unified channels of pi-ADL abstractions from Trace-Log file
To inform the correctness of data exchange between pi-SP and ASP from log files
Example questions/ indicators
Does human user interaction demonstrate the required execution flow by the respective pi-ADL specification of RAD models?
Does the execution flow within a specific process activity depict a successful interaction with specific grid services?
Are all defined abstraction and utility functions enacted and executed in the required flow?
Has proxy router enabled process activities to access specific grid services?

Figure 5: *Dynamic Validation using the Concern-based Approach*

In general, the execution behaviour of the *Submission* process has been dynamically validated and revealed the correct behaviour. This reflects on the suitability of pi-ADL as an execution language to be used in the BPMSOA framework to enact and execute a business process using application domain specific grid services (Khan, 2009). In addition, the pi-ADL specifications for RADs are still not complete and require further investigation since they currently cannot handle errors, exceptions, and are lacking in automating the proper identification of resources required for inter-role interactions at run-time.

6. ANALYSIS OF THE RESEARCH OUTCOMES

In this section, we briefly reflect on the BPMSOA's layered architecture, the use of metamodels, pi-ADL, and the respective model mappings at different layers.

- Business process models can be enacted in GSOA environments using the BPMSOA framework as demonstrated using the *Submission* process. The BPMSOA framework uses special purpose designed instantiation and application processes which facilitate the enactment of a role-based business process model into the GSOA environment via intermediary model translations. The orthogonal layers of the BPMSOA framework can be used to manage intermediary translated models as artefacts which can be reused for later process model enactment.
- The above characteristic suggests that the BPMSOA framework is closely aligned with the MDE approach as each layer generates a particular model of a given business process in accordance with its respective metamodels. Further, a top-down translation of Platform Independent Models (PIM) e.g. RAD, to Platform Specific Models (PSM) e.g. pi-ADL and its enactment is facilitated with a structured process. In addition, each associated metamodel has specific perspectives which capture different model dimensions such as structural, flow, non-functional, etc.
- There is a direct relationship between the particular instances of the BPMSOA metamodels, where each metamodel instance directs its associated layered component to perform specific model translation steps. For example, in (Khan 2009), the role-based *α-Metamodel* (i.e. at the *Generalisation* layer) is defined using the meta-concepts introduced in the *Process Modelling Metamodel* (i.e. at the *Process Modelling* layer). Similarly, the role-based instances of the *τ-Metamodel* and the *χ-Metamodel* (i.e. at the *Transformation* layer) are based on the concepts introduced in the role-based *α-Metamodel.* Further, particular instances of these associated metamodels can be used to capture a particular business process modelling approach such as role-based, etc. More instances of these metamodels allow BPMSOA to deal with

different process modelling approaches, and hence this suggests that BPMSOA can be a generalised framework. In addition, each metamodel is concerned with different perspectives (e.g. *α-Metamodel* consists of *Structural, Flow* and *Interaction* perspectives), where each is realised using a number of elements (e.g. *Structural* perspective of α-metamodel consists of *Composition, Flow* and *Interaction* elements). The modular design of these metamodels is flexible enough to accommodate new perspectives (for example, a *Functional* perspective) and elements (such as *Control* elements, etc.), which suggests that BPMSOA can also be an extensible framework.

- The extent to which the intermediary translations can be automated in the BPMSOA framework is subject to generic mappings between different technological spaces such as role-based business process modelling languages, XML and pi-ADL. The application of the role-based BPMSOA framework on the *Submission* process of the DL domain has yielded that it is not straightforward to have such generic mappings because it requires bespoke design of the intermediary model representations and, in particular, the pi-ADL specifications associated with the dynamic behaviour of different business process modelling constructs. These limitations can be attributed to the restricted capability of these technological spaces (e.g. the business process modelling language space and architectural description languages such as pi-ADL) to be mapped and translated fully from one to the other. Hence, this reveals that the extent to which the BPMSOA framework can bridge the gap between business process models and service oriented environments depends on the capability to correctly map and translate business process models into their respective XML and pi-ADL specifications.
- Furthermore, the limitation of the translation automation may also be attributed to coarsely defined process models, ad-hoc, randomly presented resources or to data elements, and the associated human user responsibilities within the specific roles of the business process models. In addition, these limitations are not straightforward to identify automatically by intermediary translation tools. In contrast, humans are capable of handling the above limitations in real time and can make non-deterministic decisions to understand the semantics of specific process activities and the need for the required resources/data elements of particular business process models. However, software tools are constrained without human support in terms of correctly identifying and translating the syntactic representations and the semantic behaviour of coarsely grained process models into executable code. Hence, this suggests that role-based business process models can be enacted in GSOA environment but the translations for these process models into executable programs must be partially facilitated by programmers to aid undefined or coarsely grained execution semantics with additional resource or data requirements (Khan, 2009).
- The use of pi-ADL and its associated ArchWare tools play a major contribution in bridging the gap between business process models and GSOA environments. Although pi-ADL is considered a block-structured architectural description language, its *channel unification* and *mobility* features enable it to also capture graph-based models such as RAD, etc. For example, RAD modelling constructs are represented as abstractions using pi-ADL, which can embody the required programming expressions, statements, etc. related to the behavioural requirement for that particular modelling construct. When these abstractions are enacted, execution control can be passed from one abstraction instance to another using the unified and mobile channels between these instances through the *send* and *receive* action prefixes. These features enable the pi-ADL specification of a business process model to adhere to the specified control flow (Khan, 2009). However, the use of channel unification and mobility may result in erroneous process execution and/or halt process execution due to input/output (I/O) blocking (i.e. synchronous behaviour of *send* and *receive* prefixes), if improper channel unification is performed. As a consequence, this raises the need to incorporate advanced debugging facilities in order to identify possible erroneous channel unification to be applied for correct utilisation. Also, new design mechanisms can be investigated to facilitate asynchronous channel based communication in order to avoid limitations which can be caused due to I/O blocking prefixes.
- The pi-ADL specification of the *Submission* process (and in particular loop handling) shows that pi-ADL can execute the graph-based process structures using its channel based mobile communication characteristics. For example, the BPMSOA pi-ADL *iterative_flow* and *abst_comp* abstractions are activated using such characteristics. Thus, *in the absence of such characteristics, it will be difficult to achieve such graph-based (e.g. loops) behaviour* let alone designing pi-ADL specifications for RAD process models (Khan, 2009).
- Human roles or activities (i.e. activities with human intervention) can be simulated and executed in pi-ADL using a terminal node. Such roles and activities need to be distinguished from the outset in the *Process Modelling* and *Generalisation* layers since the BPMSOA and its respective role-based pi-ADL specifications do not cater for such a need unless it is identified in the above mentioned two layers.
- The application of the *Submission* process using the BPMSOA framework leads us to strongly conclude that pi-ADL can capture different dimensions (as envisaged in the MDE approach (Stuart, 2002)) of the original business process model such as platform specific executable specifications which can also include:
  - a subject area dimension such as human user based roles (e.g. the *Author* and the *Editor* roles in the *Submission* process), or system supported activities (e.g. in the *Submission* process, performing initial evaluation for a submitted article); and,

    - an aspect dimension such as process flow (within a role or process model) which may include sequence, concurrency, iteration, distribution, information and data. This enables programmers to program both block-structured as well as graph-based business process models in pi-ADL.
- The application of the separation of concerns design principle has yielded a logical separation between sub components of BPMSOA layers. For example, the *proxy router* component enables pi-ADL based process activities to communicate with respective grid services without interfering with the associated pi-ADL based process execution (Khan, 2009; Khan, et. al, 2008b). Furthermore, the design of the *proxy router* sub-components can be extended to introduce a Universal Description Discovery and Integration (UDDI) like mechanism and can also access the required functionality from other IT applications, e.g. web-services, P2P, etc.
- The above model of enacting a business process using the BPMSOA framework further suggests its suitability for process agility (Khan, 2009). For example, while the execution behaviour of RAD modelling activities may not be determined using its respective RAD model, this execution behaviour is acquired from the respective application domain specific grid services. This approach demonstrates that the actual process workflow is translated into respective pi-ADL specifications and is executed in the ArchWare Virtual Machine (ADLVM) (ArchWare, 2005). However, the behavioural functionality of the executing process activities is acquired from the application domain specific grid services using the *proxy router* component. On the one hand, a change in the process design topology (i.e. its workflow or structure) is not affected by the behavioural functionality of the respective process activities since it can be accessed from specific application services. On the other hand, binding a new application domain specific grid service (i.e. with different functionality) with the specific executing process activity can also accomodate the required behavioural changes.

In addition, the following limitations have been revealed which will be dealt as future research work:

- *The absence of an application domain specific resource data model can make it difficult to automatically specify a business process in pi-ADL,* because it is not possible to automatically determine from the given RAD model of a business process the data elements or resources which need to be exchanged during inter-role interactions, for example, an article or manuscript in the case of the *Submission* process. This indicates that an application domain specific data model needs to be introduced in the BPMSOA framework that can be associated with business process representations at different layers, which can be considered as a step towards handling this shortcoming.
- *Designing and customising the pi-ADL specification for complex RAD process models is not a straightforward task.* For example, as the level of control flow complexity (e.g. with nested gateways, overlapping loops, loops with hanging thread fragments and replicate refinement) increases in a RAD business process model, this may require refinements in the current RAD pi-ADL specifications. For example, in a RAD model a *replicate refinement* (indefinite part-refinement) can be represented. This however, cannot be accommodated by the current RAD pi-ADL specifications due to the need to specify the required number of part threads with their constructs and channel unifications. Hence, more case studies with complex control flows need to be conducted in order to find remedies for such shortcomings.
- *Errors and Exceptions can lead to the unexpected execution behaviour of a given business process.* The current pi-ADL specifications for RAD models do not cover exception and error handling, mainly due to the non-existence of any asynchronous channel communication in pi-ADL. This indicates that the current RAD pi-ADL specification needs to incorporate additional mechanisms to handle errors and exceptions by incorporating asynchronous communication design.

7. CONCLUSION

In conclusion and despite the above issues discussed in section 6, the evaluation outcomes reflect on the *usefulness* of BPMSOA as a framework that has been instantiated using the role-based process modelling paradigm and in particular using RADs as a step towards bridging the gap between role-based business process models and service oriented environments. In particular, the use of pi-ADL as a process execution language demonstrates its capability to specify, enact and execute complex business process models. We used a simple case study from digital libraries domain to establish a foundation for conducting the experiments. The same approach (i.e. by using the BPMSOA framework) could be adopted for other application domains (e.g. finance, etc.) to integrate business process activities with grid utility services. However, the RAD pi-ADL specifications need further extensions (for example, handling run-time errors) and refinements (for example, static parametric requirements to invoke respective grid services), which are anticipated to increase the *usability* of the BPMSOA framework when validated using different other process modelling paradigms and complex designed business processes.

REFERENCES

Giaglis, G. M., (2001), A Taxonomy of Business Process Modelling and Information Systems Modelling Techniques, The International Journal of Flexible Manufacturing Systems, vol. 13, pp. 209-228.

Curtis, B., (1992), Process Modelling, Communications of the ACM, 35(9), pp. 75-90.

Aguilar-Saven, R. S., (2003), Business process modelling: Review and Framework, International Journal of Production Economics, 90(2), pp. 129-149.

Havey, M., (2005), Essential Business Process Modelling, ISBN. 0-596-00843-0, O'Reilly Publishers.

Kurtev, I., Bezivin, J. and Aksit, M., (2002), Technological Spaces: an Initial Appraisal, Federated Conferences: Cooperative Information Systems (CoopIS) - International Symposium of Distributed Objects and Applications (DOA), Industrial Track, Irvine, California.

Krafzig, D., Banke, K. and Slama, D., (2004), Enterprise SOA: Service-Oriented Architecture Best Practices, ISBN. 0-13-146575-9, Prentice Hall PTR.

BEA Systems, (2007), The Integration Journey - a Field Guide to Enterprise Integration for SOA [online], *BEA* White Paper, Available from: http://contact2.bea.com/bea/www/soa_mom/mom3.jsp?PC=56TU3GXXWPBE [Accessed 10/07/2007].

Breivold, H.P. and Larsson, M., (2007), Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles, 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 13-20.

Oquendo, F., (2004), pi-ADL: An Architecture Description Language based on the Higher-Order Typed pi-Calculus for specifying Dynamic and Mobile Software Architectures, ACM Software Engineering Notes, 29(4).

Oquendo, F., (2005), Tutorial on ArchWare ADL - Version 2 - Pi-ADL Tutorial - Project Deliverable D1.9 [online], Available from: http://www.arch-ware.org/ [Accessed 15/05/2006].

Milner, R., (1999), *Communicating and mobile systems: the pi calculus*, ISBN. 052164320, Cambridge University Press.

NGG - Next Generation Grids, (2006), Future for European Grids: GRIDs and Service Oriented Knowledge Utilities: Vision and Research Directions 2010 and Beyond [online], NGG Expert Report, Available from: URL Access: ftp://ftp.cordis.lu/pub/ist/docs/grids/ngg3_eg_final.pdf [Accessed 15/6/2008].

SOKU, (2006), From Grids to Service-Oriented Knowledge Utilities: A critical infrastructure for business and the citizens in the knowledge society [online], EC Report ISBN: 92-79-02116-8, Available from: ftp://ftp.cordis.europa.eu/pub/ist/docs/grids/soku-brochure_en.pdf.

Sotomayor, B., (2005), The Globus Toolkit 4.0 [online], Available from: http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html [Accessed 14/12/2007].

Manset, D., Verjus, H., McClatchey, R. and Oquendo, F., (2006). A Formal Architecture-centric model-driven approach for the automatic generation of Grid applications, *ICEIS06,* Paphos, Cyprus.

Stuart, K., (2002), Model Driven Engineering, In Book: Integrated Formal Methods - Lecture Notes in Computer Science, Springer Berlin/Heidelberg*,* ISBN: 978-3-540-43703-1, vol. 2335/2002, pp. 286-298.

Fondement, F. and Silaghi, R., (2004), Defining Model Driven Engineering Processes [online], 3rd Workshop in Software Model Engineering/the 7th International Conference on the UML (WiSME@UML2004), Lisbon, Portugal. Available from: http://www.metamodel.com/wisme-2004/papers.html [Accessed 04/02/2008].

Khan, Z. A., (2009), Bridging the gap between business process models and service oriented architectures with reference to the Grid environment, PhD Thesis, Bristol Institute of Technology, University of the West of England, Bristol, UK. British Library EThOS Persistent ID: uk.bl.ethos.501099

Ould, M., (1995), Business Processes: Modelling and Analysis for Re-engineering and Improvement, ISBN. 0-471-95352-0, Wiley.

Khan, Z.A., Odeh, M. and McClatchey, R., (2006a), Digital Libraries: From Process Modelling to Grid-based Service Oriented Architecture, Proceedings of the 2nd Int. Conference on Information & Communication Technologies from Theory to Application (ICTTA'06), Damascus, Syria, vol. 1, pp. 280-285.

Odeh, M., Beeson, I., Green, S. and Sa, J., (2002), Modelling Processes Using RAD and UML Activity Diagrams: an Exploratory Study, The 3rd International Arab Conference on Information Technology, ACIT2002, Doha, Qatar.

Odeh, M. and Kamm, R., (2003), Bridging the Gap between Business Models and System Models, Information and Software Technology, 45(15), pp. 1053-1060.

Martinez-Garcia, A. I. and Mendez-Olague, R., (2002), Integrating Process Modeling and Simulation Through Reusable Models in XML, In Proceedings of the Summer Computer Simulation Conference 2002, The Society for Modeling and Simulation International-SCS, pp. 452-460.

Badica, C. and Fox, C., (2001), Business Process Modelling Using Process Algebras, *OR'43 (BPCME'01)*, Bath, UK.

Arkin, A., (2002), Business Process Modelling Language [online], BPMI.org, Available from: http://xml.coverpages.org/BPML-2002.pdf [Accessed 11/08/2007].

Juric, M. B., Mathew, B. and Sarang, P., (2004), Business Process Execution Language for Web Services, ISBN. 1-904811-18-3, PACKT Publishing.

Blanvalet, S., Bolie, J. and et. al., (2006), BPEL Cookbook: Best Practices for SOA-based Integration and Composite Applications Development, ISBN. 1-904811-33-7, PACKT Publishing.

The OWLService Coalition, (2005), OWL-S: Semantic Markup for Web Services [online], Available from: http://www.daml.org/services/ [Accessed 01/10/2005].

Guo, L., Chen-Burger, Y.H. and Roberston, D., (2004), Mapping a Business Process Model to a Semantic Web Service Model, Proceedings of the Third International Conference on Web Services, ICWS 2004, San Diego, California, USA.

White, S.A., (2005), Using BPMN to Model a BPEL Process, *BPTrend*.

Owen, M. and Raj, J., (2007), BPMN and Business Process Management: Introduction to the New Business Process Modelling Standard [online], Available from: http://www.bpmn.org/Documents/6AD5D16960.BPMN_and_BPM.pdf [Accessed 17/07/2007].

White, S.A., (2007), Using BPMN to Model a BPEL Process [online], Available from: http://www.bpmn.org/ [Accessed 10/07/2007].

Ouyang, C., Aalst, W. M., Dumas, M. and Hofstede, A. H., (2006), Translating BPMN to BPEL [online], Available from: http://eprints.qut.edu.au/archive/00003615/ [Accessed 10/08/2007].

Recker, J. and Mendling, J., (2006), On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modelling Languages, CAiSE 2006 Workshop Proceedings - Eleventh International Workshop on Exploring Modelling Methods in Systems Analysis and Design, pp. 521-532.

BPMN, (2007), Business Process Modelling Notation (BPMN) [online], Available from: http://www.bpmn.org/ [Accessed 10/07/2007].

Shapiro, R., (2002), A comparison of XPDL, BPML, and BPEL4WS [online], ebPML.org, Available from: http://xml.coverpages.org/bpml.html [Accessed 10/05/2008].

Khan Z. A. and Odeh, M., (2007), A Framework for Translating RAD business process models into π-adl, in proceedings of Arab Conference of IT (ACIT2007), Syria.

Khan Z. A. and Odeh, M., (2008a), Business Process Modelling: Coarse to Fine Grain Mapping Using Metamodels, Proceedings of IASTED Software Engineering, Innsbruck, Austria.

Khan Z. A. and Odeh, M., (2008b), The BPMSOA: Linking Utility Grid Services with Executable Business Processes using the p-ADL based Proxy Router, Proceedings of the 3nd Int. Conference on Information & Communication Technologies from Theory to Application (ICTTA'08), Damascus, Syria.

Khan Z. A., Odeh, M., Solomonides, T. and Oquendo, F., (2008c), The BPMSOA: evaluating the enactment of a business process using application domain specific grid services, Proceedings of the 10th International Conference on Information Integration and Web-based Applications Services (iiWAS2008), ISBN 978-1-60558-349-5, p.p. 232-239. Linz, Austria.

Sommerville, I. (2006), "Software Engineering", ISBN. 9780321313799, Addison-Wesley.

ArchWare, (2005), ArchWare EmptyStore Repository [online], Available from: http://intranet.cs.man.ac.uk/ipg/research/projects/AW/ [Accessed 26/03/2008].

Harrison-Broninski, K., (2007), A Role-Based Approach To Business Process Management [online], Accessed from: http://human-interaction-management.info.

RADModeller, (2006), RADModeller: Instream Tools [online], Available from: http://www.instream.co.uk/radmodeller.html [Accessed 10/09/2006].

Kotonya, G. and Sommerville, I., (2002), Requirements Engineering: Processes and Techniques, ISBN. 0-471-97208-8, John Wiley & Sons. Inc.

Blythe, J., Deelman, E., Gil, Y., (2004), “Automatically Composed Workflows for Grid Environments” IEEE Intelligent Systems July/August 2004.

Brandic, I., Pllana, S., and Benkner, S., (2008), "Specification, Planning, and execution of QoS-aware Grid workflows within the Amadeus environment", CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE, Wiley InterScience, 20(4), p.p. 331-345.

A-WARE project: EC IST FP6 project contract number: IST-05-034545, June 2006 – 2008.

Curcin, V., Ghanem, M., (2008), "Scientific workflow systems – can one size fit all?", In Proceedings of the 4th Cairo International Biomedical Engineering Conference, 2008, CIBEC 2008. 18-20 December 2008. IEEE, ISBN 978-1-4244-2695-9.

Pllana, S., Fahringer, T., Testori, J., Benkner, S., and Brandic, I., (2004), "Towards an UML Based Graphical Representation of Grid Workflow Applications", In Proceedings of European Across Grids Conference 2004, pp.149~158.

Dornemann, T., Friese, T., Herdt, S., Juhnke, E., and Freisleben, B., (2007), "Grid Workflow Modelling Using Grid-Specific BPEL Extensions, In Proceedings of German e-Science 2007, p.p 1-9.

Yu J., and Buyya, R., (2006), "A Taxonomy of Workflow Management Systems for Grid Computing", Journal of Grid Computing, Sept. 2005, 3(3-4), p.p. 171-200. Springer Science Business Media B.V., NY, USA,

Dehnert J., and Van Der Aalst, W.M.P., (2008), "Bridging the gap between business models and workflow specifications", International Journal of Cooperative Information Systems.

Friese, T., Smith, M., Freisleben, B., (2006), "GDT: A Toolkit for Grid Service Development", Proceedings of NODe/GSEM, p.p 131-147, Erfurt, Germany.

Deelman, E., Gannon, D., Shields, M., Taylor, I., (2009), "Workflows and e-Science: An overview of workflow system features and capabilities", Journal of Future Generation Computer Systems 25(5), p.p. 528-540.

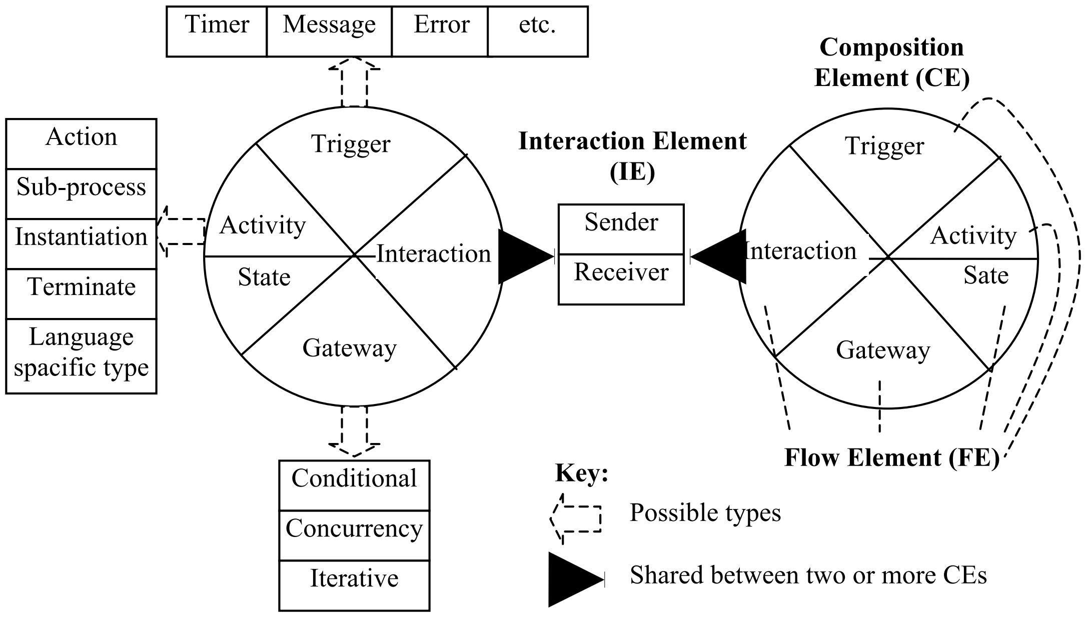
Shirasuna, S., (2011), "XBaya: A graphical workflow composer for web services", URL Access: http://www.extreme.indiana.edu/xgws/xbaya/#introduction Last Accessed: 26 April 2011.

Wassermann, B. and Emmerich, W. and Butchart, B. and Cameron, N. and Chen, L. and Patel, J., (2007), "Sedna: a BPEL-based environment for visual scientific workflow modelling". In: Taylor, I.J. and Deelman, E. and Gannon, D.B. and Shields, M., (eds.) Workflows for e-Science, ISBN 9781846285196, DOI: 5620, p.p. 428-449, London, UK.

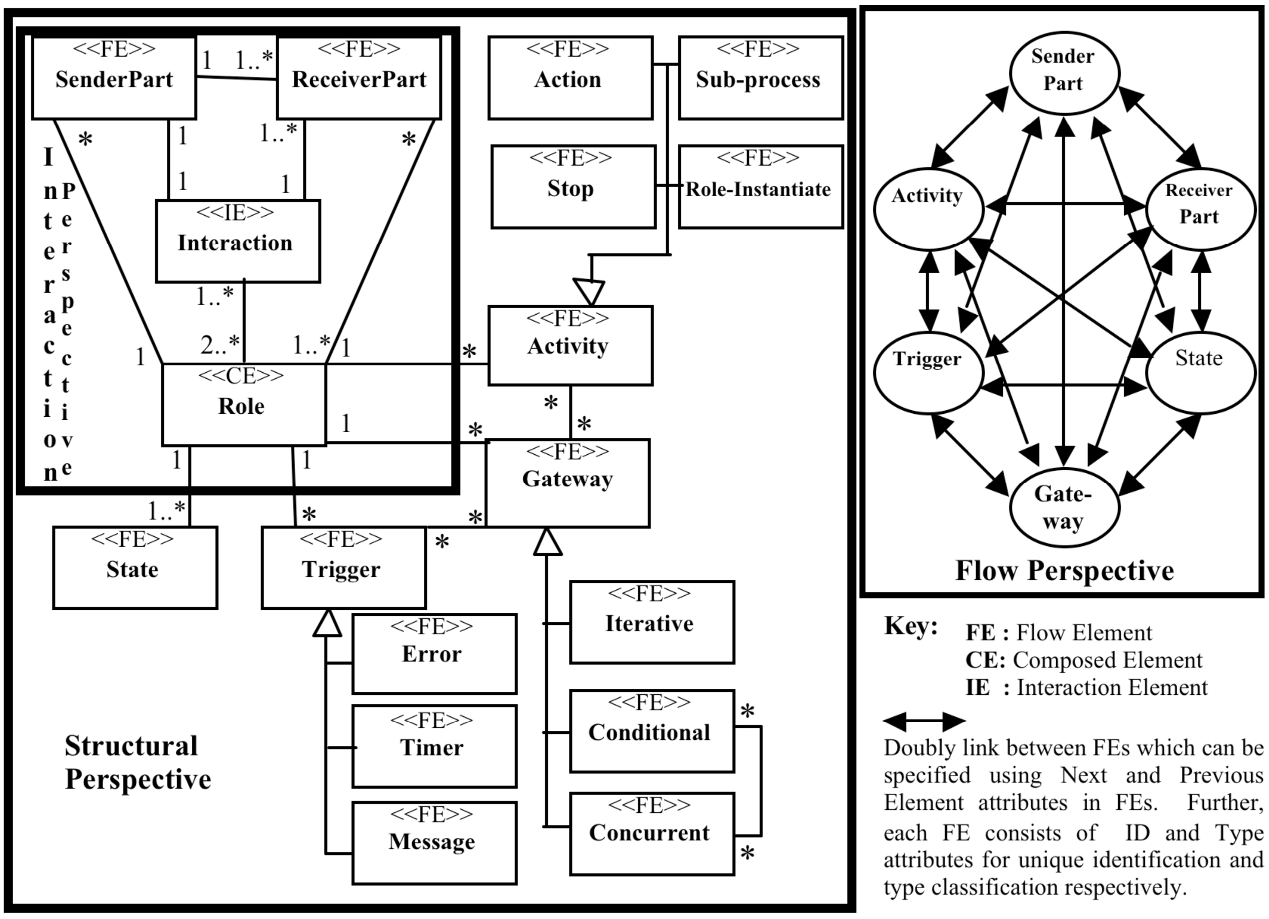
OMII-BPEL: URL Access: http://sse.cs.ucl.ac.uk/omii-bpel/docs/index.html Last Accessed: 28 April 2011.

ANNEX – I: BPMSOA FRAMEWORK: STRUCTURES OF THE ROLE-BASED METAMODELS

1. A generic role-based meta-model (Khan, 2009; Khan, et. al, 2008c)



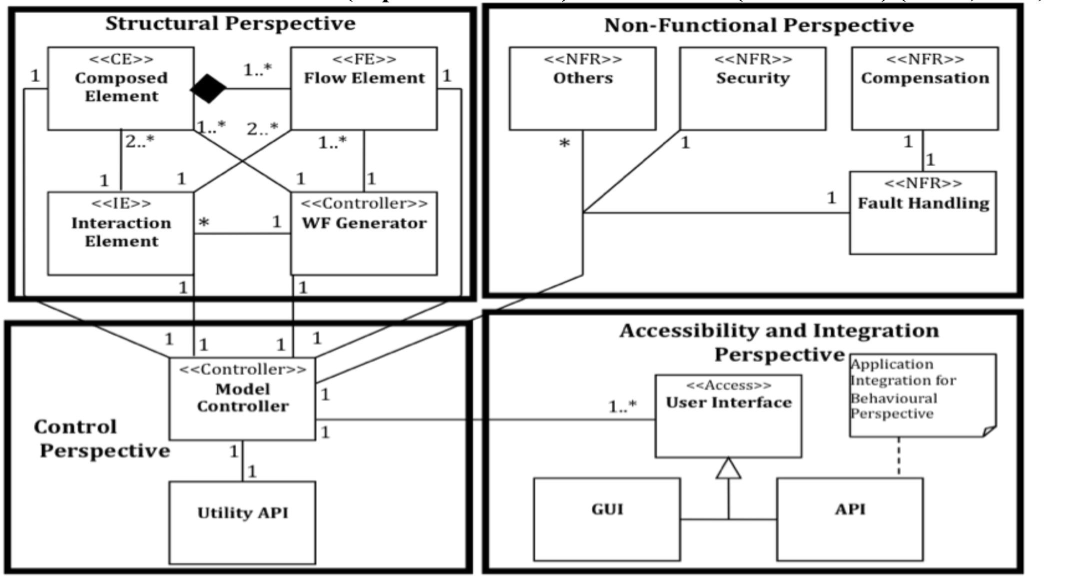
2. Structure of the Role-based (in particular RAD) α-Metamodel (Khan, 2009; Khan, et. al, 2008c)



Key: **FE :** Flow Element
CE: Composed Element
IE : Interaction Element

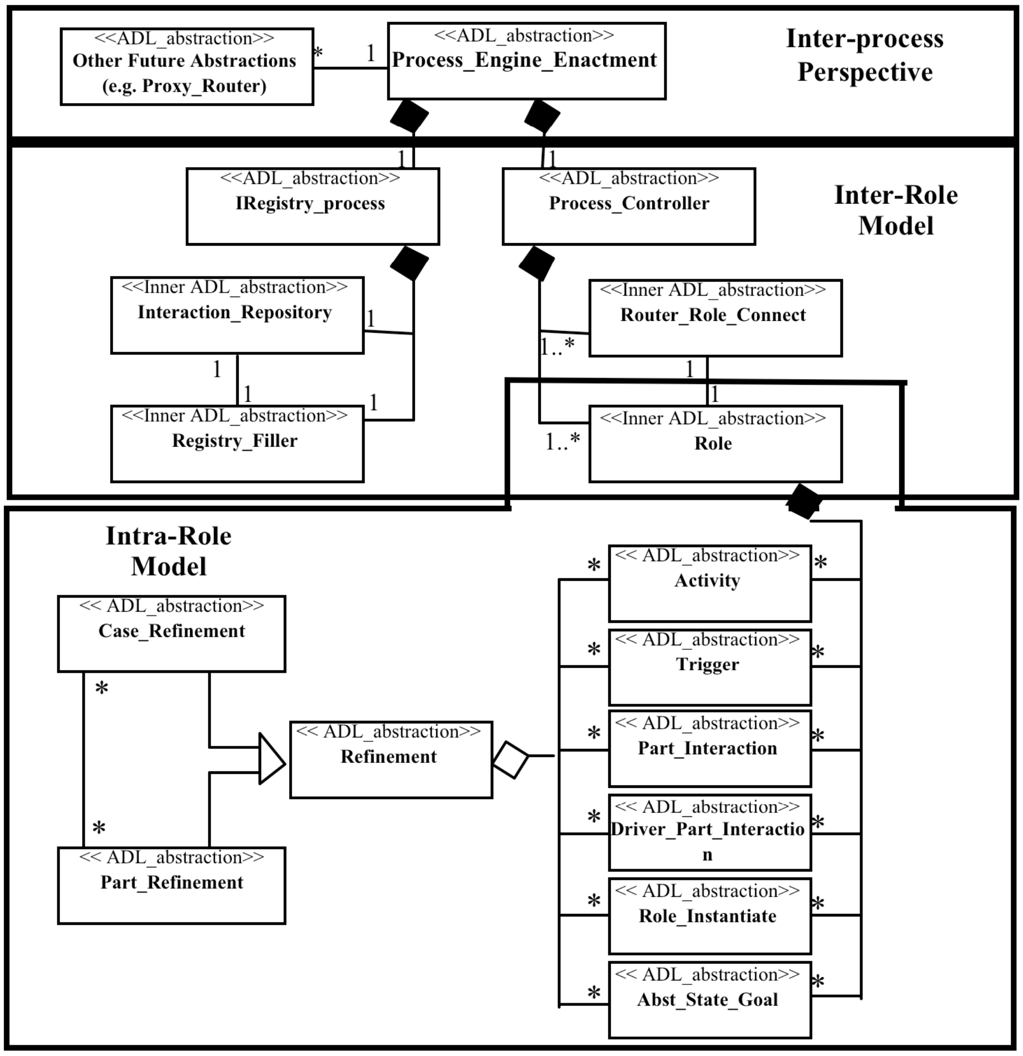
Doubly link between FEs which can be specified using Next and Previous Element attributes in FEs. Further, each FE consists of ID and Type attributes for unique identification and type classification respectively.

3. Structure of the Role-based (in particular RAD) τ-Metamodel (UML model) (Khan, 2009; Khan, et. al, 2008c)


Structural Perspective
<<CE>> Composed Element
<<FE>> Flow Element
<<IE>> Interaction Element
<<Controller>> WF Generator
Non-Functional Perspective
<<NFR>> Others
<<NFR>> Security
<<NFR>> Compensation
<<NFR>> Fault Handling
Control Perspective
<<Controller>> Model Controller
Utility API
Accessibility and Integration Perspective
<<Access>> User Interface
Application Integration for Behavioural Perspective
GUI
API
1..*
2..*
1
*

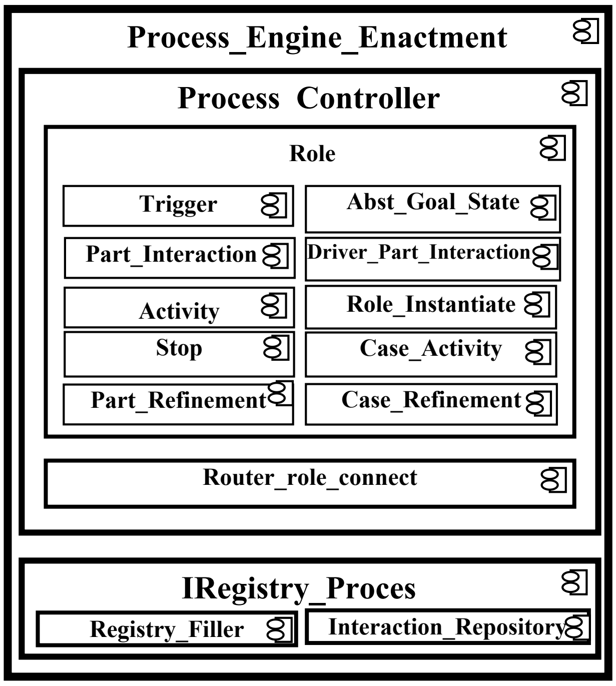
Key:
CE : Composed Element, **FE :** Flow Element, **IE :** Interaction Element, **API :** Application Programming Interface, **NFR:** Non-Functional Requirement, **GUI:** Graphical User Interface, **WF:** Work Flow

4. Structure of the Role-based (in particular RAD) χ-Metamodel (Khan, 2009; Khan, et. al, 2007)


Inter-process Perspective
<<ADL_abstraction>> Other Future Abstractions (e.g. Proxy_Router)
<<ADL_abstraction>> Process_Engine_Enactment
Inter-Role Model
<<ADL_abstraction>> IRegistry_process
<<ADL_abstraction>> Process_Controller
<<Inner ADL_abstraction>> Interaction_Repository
<<Inner ADL_abstraction>> Router_Role_Connect
<<Inner ADL_abstraction>> Registry_Filler
<<Inner ADL_abstraction>> Role
Intra-Role Model
<< ADL_abstraction>> Case_Refinement
<< ADL_abstraction>> Part_Refinement
<< ADL_abstraction>> Refinement
<< ADL_abstraction>> Activity
<< ADL_abstraction>> Trigger
<< ADL_abstraction>> Part_Interaction
<< ADL_abstraction>> Driver_Part_Interaction
<< ADL_abstraction>> Role_Instantiate
<< ADL_abstraction>> Abst_State_Goal
1
1..*
*

ANNEX – II: RAD PI-ADL SPECIFICATIONS: AN EXAMPLE

1. New Pi-ADL abstractions for RAD (Khan, 2009)



2. An example: pi-ADL code of IRegistry_Process abstraction for Interaction Repository Model (Khan, 2009)

Like an SOA model, in RAD pi-ADL specifications, the *Interaction Registry* model (Khan, 2009; Khan, et. al, 2007) enables two RAD roles to interact using a driver part and a receiver part interaction using a central registry mechanism. From RAD pi-ADL specifications (Khan, 2009), Figure B.1 (b) depicts the sequence of activities and interaction between different newly defined pi-ADL sub-abstractions to implement an interaction between two roles as depicted in Figure B.1 (a). For example, in Figure B.1 (b) the following steps are executed in sequence to perform a part interaction activity: (1) Likewise to a service provider in an SOA model, all roles register one dedicated mobile channel with the *Interaction Registry* component; (2) Similarly, all receiver part interactions inside a role register one dedicated mobile channel to *Part Registry* of the role; (3) Driver part interaction sends resource (e.g. message) to the associated instance of *Router Role Connect* abstraction which in steps (4) and (5) gets the registered mobile channel of the receiver role of the part interaction from the *Interaction Registry*; (6) Then the *Router Role Connect* forwards the resource on the received mobile channel of the receiving role; After receiving the resource, in steps (7) and (8), the receiving role gets the registered mobile channel of the receiving part interaction from the *Part Registry*; (9) Then this resource is forwarded to the receiving part interaction on the received mobile channel of the part interaction.

The *IRegistry_Process* abstraction is the implementation of the *Interaction Registry* model using the pi-ADL. It consists of two sub abstractions: the *Interaction_Registry* and the *Registry_Filler.* The instance of *Registry_Filler* abstraction utilises the addresses and names of roles and interaction activities (i.e. Interaction Elements (IE) from the *τ-Metamodel*) to populate the *Interaction_Registry (also referred as Interaction_Repository).* This results in binding the addresses of both the sender and the receiver part interactions with the corresponding roles in the *Interaction Registry* at the start of the execution of a business process. The *Interaction_Registry* abstraction provides many sub abstractions which are used to manage the *Interaction Registry* (i.e. store, extract and remove the addresses and names of roles and interactions activities).

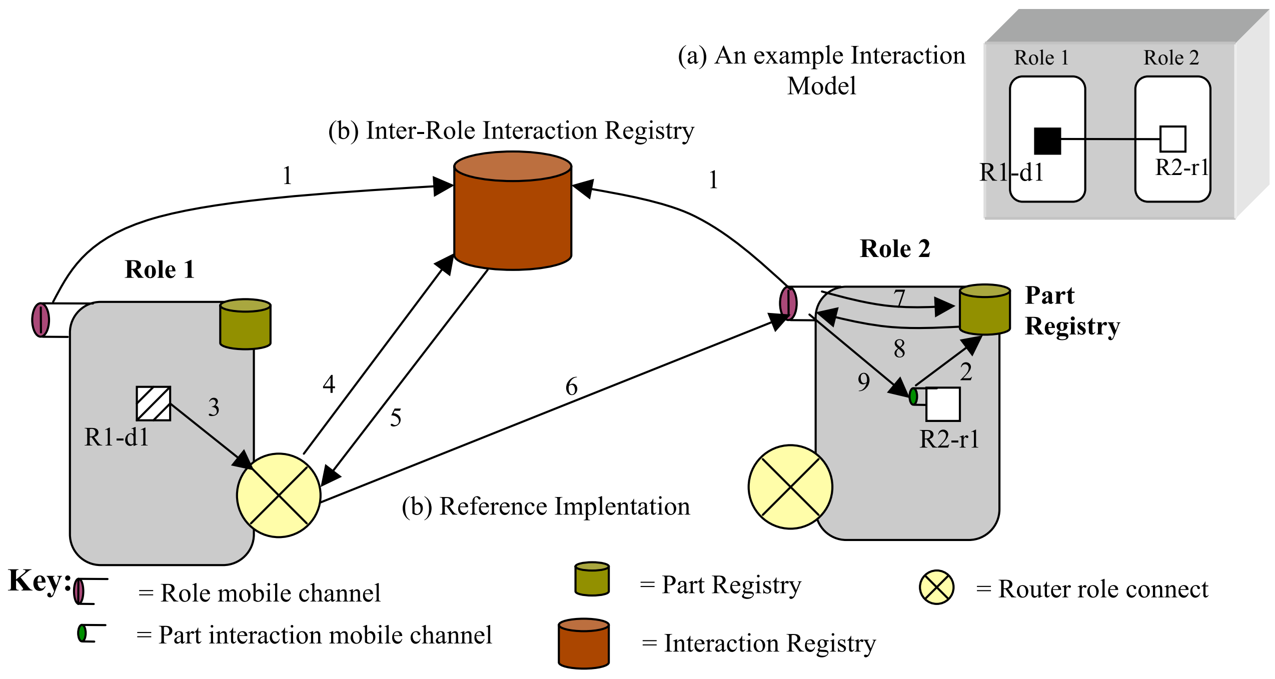


Figure B.1 *Interaction Registry working model (Khan, 2009)*
The pi-ADL code for the above *Interaction Repository* model (Figure B.1 (b)) is detailed below (anything after '!' is considered as comments by the ADL Virtual Machine).

```
value IRegistry_process = abstraction();{
        value tuple_Registry = sequence(dummy_IR_tuple)
        value t_registry = location(tuple_Registry)
        value address_registry = sequence(dummy_Rol_Address)
        value r_registry = location(address_registry)
        value mobile_IRtuple_channel_rec = connection(IR_Tuple)
        value mobile_IRtuple_channel_rec1 = connection(IR_Tuple)
        value send_tuple = connection (IR_Tuple)
        value tuple_rec_channel = connection(connection[IR_Tuple])
        value tuple_rec_channel1 = connection(connection[IR_Tuple])
        value role_channel = connection(Role_Address)
        value role_name_channel = connection(string)
        value role_add_channel = connection(connection[Trans_Element])
        value rem_add_mchannel = connection(Role_Address)
        value addr_rec_channel1 = connection(connection[Role_Address])

        !Following abstraction implements Interaction Repository
        value Interaction_Repository = abstraction();{
                value tuple_channel = connection(connection[IR_Tuple])
                value mobile_IRtuple_channel = connection(IR_Tuple)
                value dummy_tuple = location(dummy_IR_tuple)
                tuple_Registry excluding dummy_IR_tuple
                address_registry excluding dummy_Rol_Address
                via tuple_channel send mobile_IRtuple_channel
                via tuple_rec_channel send mobile_IRtuple_channel_rec
                via tuple_rec_channel1 send mobile_IRtuple_channel_rec1

                value receive_and_add_tuple_values = abstraction();{replicate{
                        via mobile_IRtuple_channel receive IR_entry : IR_Tuple
                        tuple_Registry including IR_entry
                        t_registry := tuple_Registry
                        done}
                } ! End of receive_and_add_tuple_values
```

```
value receive_and_add_role_address = abstraction();{replicate{
          via role_channel receive role_reference : Role_Address
          address_registry including role_reference
          r_registry := address_registry
          done}
} ! End of receive_and_add_role_address

value get_address_from_Registry = abstraction();{replicate{
          via role_name_channel receive role_name : string
          value dummy_address = view(name=role_name,mchannel=dummy6)
          iterate 'r_registry by r : Role_Address
                    from ra = location(dummy_address)
                    accumulate { if (r.name = 'ra.name) then r else 'ra }
          via role_add_channel send 'ra.mchannel
          done}
} ! End of get_address_from_Registry

value remove_address_from_Registry = abstraction();{
          via addr_rec_channel1 send rem_add_mchannel
          replicate{
                    via rem_add_mchannel receive rm_add : Role_Address
                    iterate 'r_registry by r : Role_Address
                              from rt = location(rm_add)
                              accumulate {if (r.name='rt.name) then r else 'rt}
                    address_registry excluding 'rt
                    r_registry := address_registry
          done}
} ! End of remove_address_from_Registry

value get_tuple_from_Registry = abstraction();{replicate{
          via mobile_IRtuple_channel_rec receive d : IR_Tuple
          dummy_tuple := d
          iterate 't_registry by e : IR_Tuple
                    from dt = location(d)
                    accumulate {
                              if ((e.drv_role='dt.drv_role) and (e.drv_entity='dt.drv_entity)) then e else 'dt
                    }
          via send_tuple send 'dt
          done}
} ! End of get_tuple_from_Registry

value remove_tuple_from_Registry = abstraction();{replicate{
          via mobile_IRtuple_channel_rec1 receive rm_tuple : IR_Tuple
          iterate 't_registry by e : IR_Tuple
                    from dt = location(rm_tuple)
                    accumulate {
                              if ((e.drv_role='dt.drv_role) and (e.drv_entity='dt.drv_entity)) then e else 'dt
                    }
          tuple_Registry excluding 'dt
          t_registry := tuple_Registry
          done}
} ! End of remove_tuple_from_Registry

compose{
          rm_tp_reg as remove_tuple_from_Registry()
and
          gt_tp_reg as get_tuple_from_Registry()
and
          rec_tp_reg as receive_and_add_tuple_values()
and
          get_add_reg as get_address_from_Registry()
```

```
        and
            rec_add as receive_and_add_role_address()
        and
            rm_ad_reg as remove_address_from_Registry()
        }
        done
    } ! End of Interaction Repository

    !Following abstraction implements Registry Filler
    value Registry_Filler = abstraction();{
        value sir_tuple_channel = connection(connection[IR_Tuple])

        value set_interaction_registry = abstraction(sir_channel:connection
                                                      [connection[IR_Tuple]]); {
            value tuple_receive = connection(IR_Tuple)
            via sir_channel receive m_channel : connection[IR_Tuple]
            replicate{
                via tuple_receive receive tuple_value : IR_Tuple
                via m_channel send tuple_value
            done}
        } ! End of set_interaction_registry

        ! Following abstraction fills IR with IR_Tuple (for Figure B.2 below)
        value send_registry_tuples = abstraction();{
            value tuple_value1 = view(drv_role="R1", drv_entity="R1-d1",
                                      rec_role="R2", rec_entity="R2-r1")
            value tuple_value2 = view(drv_role="R2", drv_entity="R2-d1",
                                      rec_role="R1", rec_entity="R1-r1")

            value tuple_send = connection(IR_Tuple)
            via tuple_send send tuple_value1
            via tuple_send send tuple_value2
            done
        } ! End of send_registry_tuples

        compose{
            sr as set_interaction_registry(sir_channel=sir_tuple_channel)
        and
            snr as send_registry_tuples() where {sr::tuple_receive unifies
                                                          snr::tuple_send}
        }
        done
    } !End of Registry Filler

    !Instantiating the Interaction_Repository and Registry_Filler abstractions
    compose{
        IR as Interaction_Repository()
    and
        RF as Registry_Filler() where {RF::sir_tuple_channel unifies IR::tuple_channel}
    }
    done
} ! End of IRegistry_Process abstraction
```

3. pi-ADL Script for an Inter-Role Interactions using Interaction Repository Model (Khan, 2009)

In order to demonstrate an inter-role interaction using *Interaction Repository (IR)* model as depicted in Figure B.1 (b), the following script shows two interactions between two roles (Role1 and Role2), where Role1 send a message to Role2 and after receiving the message Role2 sends back a message to Role1, as shown in Figure B.2. Please note that anything after '!' is considered as comments by the ADLVM.

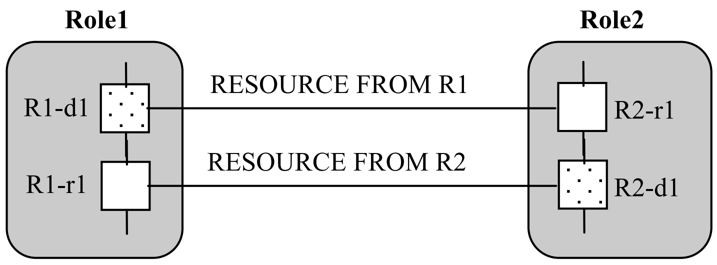


Figure B.2: *A simple RAD model depicting Inter-Role Interaction (Khan, 2009)*

```
!Terminal node channel declaration – allows Input/Output e.g. input from users, connection to sockets, etc.
value ins_value = stdio(std)

!Global Type declarations – defined in detail in (Khan, 2009)
type IR_Tuple is view[drv_role:string, drv_entity:string, rec_role:string, rec_entity:string]
type Trans_Element is view[res:string, tuple1:IR_Tuple]
type Role_Address is view[name:string, mchannel:connection[Trans_Element]]
type Part_Address is view[name:string, mchannel:connection[Trans_Element]]

!Following variables are declared to initialize different pi-ADL constructs with default values
value dummy1 = ""
value dummy2 = ""
value dummy3 = ""
value dummy4 = ""
value dummy5 = ""
value dummy6 = connection(Trans_Element)
value dummy_IR_tuple = view(drv_role=dummy1, drv_entity=dummy2, rec_role=dummy3,
                                                            rec_entity=dummy4)
value dummy_Rol_Address = view(name=dummy5, mchannel=dummy6)
value dummy_Part_Address = view(name=dummy5, mchannel=dummy6)

!Important Note!
!Here pi-ADL code of Interation Repository model will be utilised – as listed above.

!Declaration of part_interaction abstraction
value part_interaction = abstraction(rname: string, pname:string, role_channel:connection
                                [Part_Address], rm_channel:connection[Part_Address]);{
        value in_trig = connection(string)
        value out_trig = connection(string)
        value res_rec_channel = connection(Trans_Element)
        value part_address = view(name=pname, mchannel=res_rec_channel)
        via in_trig receive in_x : string
        if (in_x="true") then{
                via role_channel send part_address
                via res_rec_channel receive resource_ab : Trans_Element
                !! now also remove tuple1 from registry inside role...
                via rm_channel send part_address
                via out_trig send "true"
        }
        else{
                via out_trig send "false"
                done
        }
        done
} ! End of part_interaction abstraction

! !Declaration of driver_part_interaction abstraction
value driver_part_interaction = abstraction(drname:string, dpname:string, res:string, te_channel:
```

```
                                        connection[abstraction[]], re_channel:connection[abstraction[]]);{
        value in_trig = connection(string)
        value out_trig = connection(string)
        via in_trig receive in_x : string

        value resource_abst = abstraction();{
                value resource = res
                value resource_out_channel = connection(string)
                via resource_out_channel send resource
                done
        } ! End of resource_abst

        value Trans_envelope = abstraction();{
                value HEADER = view(drv_role=drname, drv_entity=dpname, rec_role="", rec_entity="")
                value header_out_channel = connection(IR_Tuple)
                via header_out_channel send HEADER
                done
        } ! End of Trans_envelope

        if (in_x = "true") then{
                via te_channel send Trans_envelope
                via re_channel send resource_abst
                via out_trig send "true"
                done
        }
        else{
                via out_trig send "false"
                done
        }
} ! End of driver_part_interaction abstraction

! An empty abstraction required to initialize other necessary abstractions
value dummy_te = abstraction();{
                done
} ! End of dummy_te

! Process_Controller abstraction enacts router_role_connect and roles with their associated elements
value Process_Controller = abstraction(stop_chann:connection[string]);{
        value dummy_chan = connection(string)
        value tuple_rec_channel = connection(connection[IR_Tuple])
        value tuple_rec_channel1 = connection(connection[IR_Tuple])
        value tuple_rec_value = connection(IR_Tuple)
        value r_address_channel = connection(Role_Address)
        value r_name_channel = connection(string)
        value r_maddr_channel = connection(connection[Trans_Element])
        value addr_rec_channel1 = connection(connection[Role_Address])
        via tuple_rec_channel receive m_tuple_channel : connection[IR_Tuple]
        via tuple_rec_channel1 receive m_tuple_channel1 : connection[IR_Tuple]
        via addr_rec_channel1 receive m_addr_channel1 : connection[Role_Address]

        ! Following is the router_role_connect abstraction
        value router_role_connect = abstraction(parent_role:string, rname_chan:connection[string],
                                                mchannel:connection[connection[Trans_Element]])
        {
                type IR_Tuple is view[drv_role:string, drv_entity:string, rec_role:string, rec_entity:string]
                type Rol_Address is view[name:string, mchannel:connection[abstraction[]]]
                value m_te_channel = connection(abstraction[])
                value channel_te_transf = connection(connection[abstraction[]])
                value m_re_channel = connection(abstraction[])
                value channel_re_transf = connection(connection[abstraction[]])
                value trans_env_loc = location(dummy_te)
```

```
            value trans_res_loc = location(dummy_te)
            value header_loc = location(dummy_IR_tuple)
            value resource_loc = location("resource")
            via channel_te_transf send m_te_channel
            via channel_re_transf send m_re_channel
            value receive_te = abstraction(m_tchannel:connection[abstraction[]], m_rchannel:
                                                        connection[abstraction[]]);{replicate{
                via m_tchannel receive te : abstraction[]
                via m_rchannel receive re : abstraction[]
                trans_env_loc := te
                trans_res_loc := re

                value extract_send_te = abstraction();{
                    value header_in_channel = connection(IR_Tuple)
                    value resource_in_channel = connection(string)
                    via header_in_channel receive header : IR_Tuple
                    header_loc := header
                    via resource_in_channel receive resource : string
                    resource_loc := resource
                    value trans_element = view(res=resource,tuple1=header)
                    via m_tuple_channel send header
                    via tuple_rec_value receive ttuple : IR_Tuple
                    header_loc := ttuple

                    if (ttuple.drv_role = header.drv_role) and (ttuple.drv_entity=
                                header.drv_entity) and (ttuple.rec_role="") and
                                (ttuple.rec_entity="") then {
                        via ins_value send " No entry matches inside IR , router ! 'n"
                        done
                    }
                    else{
                        via ins_value send " entry matches 'n"
                        via rname_chan send ttuple.rec_role
                        via mchannel receive rmchannel : connection[Trans_Element]
                        value resource_new = view(res=resource,tuple1=ttuple)
                        via rmchannel send resource_new
                        done
                    }
                    done
                } ! End of extract_send_te

                compose{
                    tr_en as te()
                and
                    tr_re as re()
                and
                    ex_te as extract_send_te() where { tr_en::header_out_channel unifies
                        ex_te::header_in_channel, tr_re::resource_out_channel unifies
                        ex_te::resource_in_channel}
                }
                done}
            } ! End of receive_te abstraction

            receive_te(m_tchannel=m_te_channel, m_rchannel=m_re_channel)
            done

        } ! End of router_role_connect abstraction

        !The following abstraction composes Role 1
        value Role1 = abstraction(rol_name:string, raddress:connection[Role_Address],
                                                    stop_chan:connection[string]);{
```

```
value part_registry = sequence(dummy_Part_Address)
value p_registry = location(part_registry)
value out_value = connection(string)
value loc_final = location("")
value ac1 = driver_part_interaction       ! R1-d1
value ac2 = part_interaction              ! R1-r1
value role_name = rol_name
value mchannel_role = connection(Trans_Element) !this is the role channel set in IR
value Role_Addr_IR = view(name=role_name, mchannel=mchannel_role)
value channel_te_transf = connection(connection[abstraction[]])
value channel_re_transf = connection(connection[abstraction[]])
via channel_te_transf receive m_te_channel : connection[abstraction[]]
via channel_re_transf receive m_re_channel : connection[abstraction[]]
via raddress send Role_Addr_IR
value rem_add_mpchannel = connection(Part_Address)
value part_channel = connection(Part_Address)
value part_name_channel = connection(string)
value part_add_channel = connection(connection[Trans_Element])

value res_reception = abstraction(mres_chann : connection[Trans_Element],
                part_name:connection[string], addrec_pchannel:
                connection[connection[Trans_Element]]);{
        value resource_in_channel1 = connection(string)
        value header_in_channel1 = connection(Role_Address)
        replicate{
                via mres_chann receive te_res : Trans_Element
                via part_name send te_res.tuple1.rec_entity
                via addrec_pchannel receive mchannel_part1 : connection[Trans_Element]
                via mchannel_part1 send te_res
                done
        }
} ! End of res_reception abstraction

value receive_and_add_part_address = abstraction();{replicate{
        via part_channel receive part_reference : Part_Address
        part_registry including part_reference
        p_registry := part_registry
        done}
} ! End of receive_and_add_part_address

value get_address_from_pregistry = abstraction();{replicate{
        via part_name_channel receive part_name : string
        value dummy_address = view(name=part_name,mchannel=dummy6)
        iterate 'p_registry by p : Part_Address
                from pa = location(dummy_address)
                accumulate { if (p.name = 'pa.name) then p else 'pa }
        via part_add_channel send 'pa.mchannel
        done}
} ! End of get_address_from_pregistry

value remove_address_from_pregistry = abstraction();{replicate{
        via rem_add_mpchannel receive rm_padd : Part_Address
        iterate 'p_registry by p : Part_Address
                from pt = location(rm_padd)
                accumulate {if (p.name='pt.name) then p else 'pt}
        part_registry excluding 'pt
        p_registry := part_registry
        done}
} ! End of remove_address_from_pregistry

value role_pregistry_inst = abstraction();{
```

```
                compose{
                        rr as res_reception(mres_chann=mchannel_role,
                                part_name=part_name_channel,  addrec_pchannel=part_add_channel)
                and
                        rpa as receive_and_add_part_address()
                and
                        gar as get_address_from_pregistry()
                and
                        rar as remove_address_from_pregistry()
                }
        } ! End of role_pregistry_inst

        !! Following abstraction is used to instantiate very first activity within this role
        value instantiate_activities = abstraction();{
                value inst_act = connection(string)
                via inst_act send "true"
                done
        } ! End of instantiate_activities

        !! Following abstraction links the post-state of very last activity within Role with Role
        value role_channel_comm = abstraction();{
                value in_value = connection(string)
                via in_value receive final : string
                loc_final := final
                done
        } ! End of role_channel_comm

        if (instance_x = "true") then
        {
                compose{
                        inst as instantiate_activities()
                and
                        rcc as role_channel_comm()
                and
                        rpi as role_pregistry_inst()
                and
                        c1 as ac1(drname=rol_name, dpname="R1-d1", res="RESOURCE FROM
                                R1", te_channel=m_te_channel, re_channel=m_re_channel)
                and     c2 as ac2 (rname = rol_name, pname="R1-r1", role_channel=part_channel,
                                rm_channel=rem_add_mpchannel) where {inst::inst_act unifies
                                c1::in_trig, c1::out_trig unifies c2::in_trig, c2::out_trig unifies rcc::in_value}
                }
        }
        else
        {
                via ins_value send "Role - role end false 'n"
                done
        }
} ! End of Role1


!The following abstraction composes Role 2
value Role2 = abstraction(rol_name:string, raddress:connection[Role_Address],
                                        stop_chan:connection[string]);{
        value part_registry = sequence(dummy_Part_Address)
        value p_registry = location(part_registry)
        value out_value = connection(string)
        value loc_final = location("")
        value ac1 = part_interaction                ! R2-r1
        value ac2 = driver_part_interaction         ! R2-d1
        value role_name = rol_name
value mchannel_role = connection(Trans_Element) !this is the role channel set in IR
value Role_Addr_IR = view(name=role_name, mchannel=mchannel_role)
value channel_te_transf = connection(connection[abstraction[]])
value channel_re_transf = connection(connection[abstraction[]])
via channel_te_transf receive m_te_channel : connection[abstraction[]]
via channel_re_transf receive m_re_channel : connection[abstraction[]]
via raddress send Role_Addr_IR
value rem_add_mpchannel = connection(Part_Address)
value part_channel = connection(Part_Address)
value part_name_channel = connection(string)
value part_add_channel = connection(connection[Trans_Element])

value res_reception = abstraction(mres_chann : connection[Trans_Element],
        part_name: connection[string], addrec_pchannel:connection
        [connection[Trans_Element]]);{
    value resource_in_channel1 = connection(string)
    value header_in_channel1 = connection(Role_Address)
    replicate{
        via mres_chann receive te_res : Trans_Element
        via part_name send te_res.tuple1.rec_entity
        via addrec_pchannel receive mchannel_part1 : connection[Trans_Element]
        via mchannel_part1 send te_res
        done
    }
} ! End of res_reception

value receive_and_add_part_address = abstraction();{replicate{
    via part_channel receive part_reference : Part_Address
    part_registry including part_reference
    p_registry := part_registry
    done}
} ! End of receive_and_add_part_address

value get_address_from_pregistry = abstraction();{replicate{
    via part_name_channel receive part_name : string
    value dummy_address = view(name=part_name,mchannel=dummy6)
    iterate 'p_registry by p : Part_Address
        from pa = location(dummy_address)
        accumulate { if (p.name = 'pa.name) then p else 'pa }
    via part_add_channel send 'pa.mchannel
    done}
} ! End of get_address_from_pregistry

value remove_address_from_pregistry = abstraction();{replicate{
    via rem_add_mpchannel receive rm_padd : Part_Address
    iterate 'p_registry by p : Part_Address
        from pt = location(rm_padd)
        accumulate {if (p.name='pt.name) then p else 'pt}
    part_registry excluding 'pt
    p_registry := part_registry
    done}
} ! End of remove_address_from_pregistry

value role_pregistry_inst = abstraction();{
    compose{
        rr as res_reception(mres_chann=mchannel_role, part_name=
                part_name_channel,  addrec_pchannel=part_add_channel)
    and
        rpa as receive_and_add_part_address()
    and
        gar as get_address_from_pregistry()
```

```
                and
                        rar as remove_address_from_pregistry()
                }
        } ! End of role_pregistry_inst

        !! Following abstraction is used to instantiate very first activity within this role
        value instantiate_activities = abstraction();{
                value inst_act = connection(string)
                via inst_act send "true"
                done
        } ! End of instantiate_activities

        !! Following abstraction links the post-state of very last activity within Role with Role
        value role_channel_comm = abstraction();{
                value in_value = connection(string)
                via in_value receive final : string
                loc_final := final
                done
        } ! End of role_channel_comm

        if (instance_x = "true") then
        {
                compose{
                        inst as instantiate_activities()
                and
                        rcc as role_channel_comm()
                and
                        rpi as role_pregistry_inst()
                and
                        c1 as ac1 (rname= rol_name, pname="R2-r1", role_channel=part_channel,
                                rm_channel=rem_add_mpchannel)
                and     c2 as ac2 (drname=rol_name, dpname="R2-d1", res="RESOURCE FROM
                                R2", te_channel=m_te_channel, re_channel=m_re_channel) where{ inst::inst_act
                                unifies c1::in_trig, c1::out_trig unifies c2::in_trig, c2::out_trig unifies
                                rcc::in_value}
                }
        }
        else
        {
                via ins_value send "role end false"
                done
        }
    } ! End of Role 2

    ! Following is the composition of roles and router role connector instances
    compose{
        r1 as Role1(rol_name="R1", raddress=r_address_channel, stop_chan=stop_chann)
    and
        rrc1 as router_role_connect(parent_role="R1", rname_chan=r_name_channel, mchannel=
                r_maddr_channel)
    and
        r2 as Role2(rol_name="R2", raddress=r_address_channel, stop_chan=stop_chann)
    and
        rrc2 as router_role_connect(parent_role="R2", rname_chan=r_name_channel, mchannel=
                r_maddr_channel) where{r1::channel_te_transf unifies rrc1::channel_te_transf,
                r1::channel_re_transf unifies rrc1::channel_re_transf, r2::channel_te_transf unifies
                rrc2::channel_te_transf, r2::channel_re_transf unifies rrc2::channel_re_transf  }
    }
    done
} ! End of Process_Controller abstraction
```

```
! Following is the Process_Engine_Enactment abstraction
value Process_Engine_Enactment = abstraction(); {
        value running = location(done)
        value stop_channel = connection(string)

        ! Following abstraction enacts and links IRegistry_process with Process_Controller
        value exec = abstraction(); {
                        running := compose{
                                IRP as IRegistry_process()
                        and
                                PC as Process_Controller(stop_chann=stop_channel) where {
                                        IRP::tuple_rec_channel unifies PC::tuple_rec_channel, IRP::role_channel unifies
                                        PC::r_address_channel, IRP::role_name_channel unifies PC::r_name_channel,
                                        IRP::role_add_channel unifies PC::r_maddr_channel, IRP::tuple_rec_channel1
                                        unifies PC::tuple_rec_channel1, IRP::send_tuple unifies PC::tuple_rec_value,
                                        IRP::addr_rec_channel1 unifies PC::addr_rec_channel1}
                        }
                done
        } ! End of exec abstraction

        ! Following abstraction is used to stop the process execution
        value stop_abst = abstraction(); {
                via stop_channel receive stop_signal
                if (stop_signal = "stop") then{
                        value deco = decompose 'running
                        done
                }
                else{
                        done
                }
        } ! End of stop_abst

        compose{
                ex as exec()
        and
                st as stop_abst()
        }
} ! End of Process_Engine_Enactment

! Following is the application or enactment of the above process
Process_Engine_Enactment()
```