A framework for business processes view integration

Souvik Barat¹, Vinay Kulkarni¹, Prof. D Janakiram²

¹ Tata Research Development & Design Centre, Pune, India
{souvik.barat, vinay.vkulkarni}@tcs.com
² Indian Institute of Technology Madras, Chennai, India
djram@iitm.ac.in

Abstract: The demand for integrating enterprise applications is growing as a consequence of the need to support dynamic, cross-functional and inter-organizational business processes. Traditional organization of an enterprise as a set of functionally distinct departments leads, over a time, to a set of isolated applications providing point solutions each constructed for a specific purpose. Enterprise Application Integration (EAI) discipline deals with mechanisms for putting these isolated applications together. By visualizing an enterprise application as a 3-tuple comprising of its data, service and process models, EAI problem can be visualized as a view-integration problem over data, service and process models. Semantic underpinnings of the high level notations used by Industry practice to specify the process model of enterprise applications, and tools and techniques prevalent in EAI practice are not rich enough to capture and verify the behavioral aspect of enterprise application with rigor. This limits precision of the analysis of integration requirements and support for automation. On the other hand, formal techniques like model checking and theorem proving require elaborate specifications in a language that is too low level for business applications. This work presents a framework for view-integration of process models wherein the principal objective is to analyze the process model of existing applications in the context of the desired application. The approach supported by the framework combines usability of the high level notations prevalent in industry practice and the analytical rigor due to formal techniques to verify reusability of an existing enterprise application.

We propose two integration properties, namely, compatibility property and completeness property, to formally analyze the behavioral aspect of enterprise. The proposed compatibility property addresses the discovery of reusable processes and the completeness property determines the behavioral completeness of the existing processes with respect to a desired process. We present a toolset implementing the proposed techniques to automate the verification, and a conceptual framework that can be effectively used in practice.

Keywords: EAI, Business Process, Process Integration, Semantically correct integration.

1. Introduction

Enterprises are witnessing an increased thrust on collaboration and integration of existing applications to provide value-added services across the entire supply chain. Traditionally, enterprise applications are built as point solutions with context-specific built-in assumptions hard-coded in their implementation. Enterprise Application Integration (EAI) discipline deals with the mechanism for integrating such isolated applications into a consistent whole [1]. Providing a correct enterprise application by assembling a set of existing applications is not always a simple process. Typically, enterprise applications are designed to operate in a specific context with the context-specific built-in assumptions getting encoded in their implementations. These assumptions lead to conflicts or mismatches during integration with other applications. Identification and mitigation of these conflicts during integration of isolated applications are the key challenges of EAI discipline. Also, industry practice relies solely on testing to ensure correctness of the integration. This post-facto process is time and effort intensive. This paper addresses a part of the problem of ensuring semantically-correct integration of enterprise applications wherein an enterprise application is modeled as a 3-tuple comprising of its data, service and process models, and automata synthesis techniques are used to ascertain, before deployment of the integrated application, i) if an existing
application can be reused with possible adaptation to realize the desired integrated application, and ii) if the desired integrated application can be completely realized from the existing applications with possible adaptations. Present EAI techniques provide only ‘plumbing support’ for integration of data models and for correct invocation of services in the light of the integrated data models wherein the onus of semantic correctness of integration lies entirely with the user. A little work of practical significance is seen with regard to process level integration [2]. Industry practice uses a set of modeling notations such as BPEL4WS [3], UML profile for business process [4] etc. to specify the processes of enterprise applications. However, the semantic underpinnings of these notations are not rich enough to verify the required properties of interest for process level integration and this limits precision and automation in practice. On the other hand, several approaches have been proposed based on a variety of abstract models, namely, Petri-Net [5], labeled transition system [6], CSP [7], etc., and a variety of formal techniques to analyze process model of enterprise application in the context of process level integration. Nevertheless, these approaches based on strong formal foundation have not seen wide industrial acceptance as practitioners find them too involved to use and too detailed as compared to the high level notations they are used to. Therefore, there is a need for a pragmatic approach that combines the rigor due to formal techniques and the usability and high level of abstractions due to prevalent industry practice to address the EAI problem in a comprehensive manner.

The identification of a set of appropriate applications and possible adaptations with the assurance of the completeness for a desired application is the key challenge in EAI discipline [8]. This paper presents a rigorous and pragmatic approach to address this challenge for process level integration. The process view, the behavioral aspect of the enterprise application with its built-in assumptions, is considered as the basis for establishing the desired properties of interest in our work. Two properties of interest, which we term as integration properties, namely compatibility property and completeness property, are proposed for analyzing the process views. The compatibility property determines if a process view can be fit into the desired context without any conflict and possibility of adaptation in case of mismatch, and the completeness property ascertains the completeness of a set of fitting process views with respect to a desired process and identifies the existing gap (if there exist any).

The paper presents an analyzable model, process automata, to formally represent a process view, formal underpinning of proposed properties of interest, and a set of mediation operators for adaptation of process view in conflict for a class of mismatches. A framework is presented to support for proving necessary mapping between the existing state-of-art in EAI and the proposed theoretical offering. The process automaton is based on finite state automata [9] model. The concept of simulation relation [6] and language containment [9] are adopted appropriately with the notion of refinement [10] for ascertaining the proposed properties of interest, and the mediation operators are based on the concept of refinement, abstraction and hiding. The proposed framework, combining techniques and methodologies prevalent in industry practice with rigor of formal methods, is based on the basic concept of Model Driven Architecture (MDA) [11].

The rest of this paper is organized as follows. Section 2 states the problem formulation in brief. Section 3 presents an overview of the proposed formal solution. A framework for process view integration is described in Section 4. Related work is presented in Section 5 and section 6 concludes the paper with a brief outline of future work.

2. Business Process View Integration Framework

Suppose $A_{desired}$ is the desired application that needs to be realized through maximal reuse of existing applications $A_1, A_2, ..., A_n$. The complete integration process typically spans across three phases: discovery, definition and implementation. The discovery phase ascertains whether an existing application $A_i$ fits into the context of $A_{desired}$ or not, checks for the possibility of adaptation to make it fit in case of a mismatch, and checks a set of fitting applications $\{A_1,...,A_k\}$ for complete realization of the desired application $A_{desired}$. The definition phase constructs a composite application by defining appropriate interactions of participating applications. Finally, the implementation phase implements the desired integrated application by considering the various implementation issues.

As we visualize an enterprise application as the composition of three concerns of interest, namely data, service and process [12], the EAI problem can be visualized as the integration problem over data, service and process models. Since an enterprise application is designed to operate in a specific, and typically, isolated context, the context-specific assumptions typically get built into their implementations.
and may lead to conflicts or mismatches while putting these isolated applications into the desired integrated context. These conflicts or mismatches can be identified by verifying each of the existing applications using a set of properties, which we term as integration properties.

We formulate two integration properties, namely compatibility property and completeness property. The compatibility property determines if an application can fit into the desired context without any conflict or mismatch. The completeness property ascertains if the desired application can be completely realized as a composition of the candidate applications and identifies gap if any.

This work formulates the compatibility and completeness properties for the process model of enterprise application. Typically, process model of an enterprise application comprises of three aspects, namely, behavioral aspect, interaction aspect and the execution aspect. The behavioral aspect is addressed in the discovery phase, the interaction aspect in the definition phase and the execution aspect in the implementation phase. We term the abstract specification of the behavioral aspect of an enterprise application with various encoded built-in assumptions as process view. Typically, a process view is a control flow over a set of process activities with execution-order dependencies and conditional expressions made explicit. A process activity, an atomic unit, is either a service offered/required by an enterprise application or a manual task. We introduce a notion called composition setting to analyze the behavioral completeness of a set of application applications with respect to desired integrated application. Informally, a composition setting is a configuration describing the process views of the desired application and of the existing applications that are selected for integration.

Typically, industry practice uses a set of modeling notations such as BPEL4WS, UML profile for business process etc. to specify business processes. However, the semantic underpinnings of these notations are not rich enough to formally analyze for the desired integration properties (for EAI). On the other hand, general purpose formalisms, such as FSA, Petri Net etc., support formal analysis but industry practitioners find them at too low a level and too detailed for specifying business processes. Moreover, these formalisms are not rich enough to capture various concerns of an enterprise application. Ideally, each of these formalisms can specify a specific concern of an enterprise application or a specific aspect of a concern. For example, Finite state automata, Petri-Net suited for specifying the behavioral aspect of the process model, and CSP can be used for specifying the interaction aspect. Our framework supports easy specification of application (i.e. application model) as a set of concern-specific models using high level modeling notations prevalent in industry practice. Framework uses model transformation techniques to transform these high level specifications to a suitable representation (i.e. analysis-specific model) that is amenable for rigorous analyses. The framework provides bi-directional traceability across the modeling layers shown in Fig. 1.

3. A Formal approach to Business Process View Integration

The completeness property of a process view with respect to the desired process view verifies the execution-orders of all involved process activities of the existing process view with respect to the desired
process view. The completeness property of a set of participating process view with respect to the desired process view verifies the required process activities of the desired process view are present in the participating process views and they do not violate any execution-orders within the context of their own process view. Fundamentally, the proposed integration properties are kinds of checking refinement [10] using system equivalence and their pre-order relations. We use the notion of simulation relation [6] to verify the proposed integration properties. The relations can be verified if the involved system can be visualized as a label transition system.

Different research initiatives use different formalisms, for instance Automata, Petri-Net, various process calculus, etc., to analyze various aspects of business processes [5, 13, 14, 31]. The process calculi, such as CSP, pi-calculus, etc., are designed to specify the interaction aspect of a process model rather than behavioral aspect. Thus they are more useful in the context of describing the interactions of communicating applications for an integrated application. For instance, a fundamental aspect of CSP involves the synchronization of concurrent processes on their inputs and outputs. The synchronization only occurs when two processes wishing to synchronize to send a message, are both in a sending and receiving state respectively. Hence it is primarily concerned with the specification of communicating concurrent processes, and describing complex communication and interaction patterns. But it is not suitable for describing the internals behavior of a stand-alone process [15]. As we are dealing with the behavioral aspect of stand-alone process model, the finite state automata or Petri-Net model can be considered as the formal model in our approach. Several variants of Petri-Net have been used to formalize business processes [5] and to analyze [31, 32] various properties of a business application in the context of composition. Since the scalability of Petri-Net as a modeling technique is questionable for real world scenario [15], we have chosen Finite state automata model to formally represent the behavioral aspect of the enterprise application in our approach.

As we use finite state automata, termed as process automata, for representing the process view of enterprise application, the verification of integration properties can also be visualized as the automata synthesis problem. This section provides an overview of the formal underpinnings of the proposed approach.

3.1 Process Automata

Formally, we represent a process view as a deterministic finite state automaton wherein process states are the states of automaton, process activities are the alphabets or the events, and activity flows are the state transition relations. Parallelism and synchronization between activities is addressed by flattening out the possible interleaving of process activities. The conditional expressions are also considered as the events and the transitions of the transition system. A process automaton P is a 5-tuple: P (S, E, T, s, F) where

S is a finite non empty set of process states.
E is a finite set of events, which represent the vocabulary of process activities and conditional expressions. The process activities and conditional expressions are suitably mapped onto the events. The vocabulary, and hence events, are classified into three types, namely, input events (I), output events (O), Variable (V) i.e. E = I ∪ O ∪ V.
T is a non empty set of transitions. A transition t ∈ T is defined as, s −→ e s', where s is the source state, s' is the target state and e is an event. The structural constructs between the process activities and conditional expressions are suitably mapped into the transitions relations.
s is the start state (s ∈ S). We assume there is no incoming transition to the start state.
F is the set of final states (F ∈ S). We assume that final states have no outgoing transitions.

3.2 Operators

In order to introduce various formal representations and the analysis techniques, we introduce a set of operators and functions for the proposed process automata.

Type (e)

Type function returns the type of the event e, i.e. Input, Output or Variable.
Mapping (M: E₁ → E₂)

In a real-life integration scenario, different enterprise applications may use slightly different business terminologies that reflect in different vocabularies being used to describe their process views. This necessitates a mapping to be specified between these vocabularies. Mapping describes a correspondence between a set of events E₁ of process P₁ to a set of events E₂ of process P₂.

\[ e₁ : P₁ \rightarrow e₂ : P₂ \Rightarrow \text{Event } e₁ \text{ of process } P₁ \text{ corresponds to event } e₂ \text{ of process } P₂. \]

We define a mapping function \( f_{map}(\text{event}) \), which returns either the corresponding event \( e₂ \) of \( E₂ \) for a given event \( e₁ \) of \( E₁ \) from the provided \( M \) if event exists in \( M \) or the event itself.

Restriction (P₁, P₂, M)

Given a mapping \( M \), the Restriction of a process automaton \( P₁ \) by a process automaton \( P₂ \) results in a process \( Pₐ \) that contains only those transitions of \( P₁ \) whose corresponding events are present in process \( P₂ \). Consider processes \( P₁ = (S₁, E₁, T₁, s₁^0, F₁) \), \( P₂ = (S₂, E₂, T₂, s₂^0, F₂) \) and \( Pₐ = \text{Restriction} (P₁, P₂, M) \), the Restriction operator ignores the transitions of \( P₁ \) having labels from the set \((E₁ – M(E₂))\), i.e. Restriction \((P₁, P₂, M) = \text{Ignore} (P₁, - M(E₂)))\). Given a process \( P \) and a set of events \( I \), Ignore operator computes the transitive closure graph by considering the set of transitions triggered by \( e \in I \) as epsilon moves and constructs an equivalent deterministic finite automaton using subset construction algorithm [Automata].

3.3 Formal Representation

Composition setting

We introduce the concept composition setting for reasoning the completeness property. A composition setting is a configuration describing the process views of the desired application and of the existing applications that are considered for the integration. Consider a set of participating applications \( \{A₁, A₂, \ldots Aₖ\} \) selected for the integration to achieve the desired application \( A_{desired} \). Formally, a composition setting can be described as follows: a composition setting \( CS \) is a 3-tuples \( (P₁, P₂, M) \), where

- \( P₁ \) is process automaton of the process view of desired application \( A_{desired} \).
- \( P₂ \) is a set of process automata \( \{P₁, P₂, \ldots Pₖ\} \) of process views of participating applications \( A₁, A₂, \ldots Aₖ \) respectively.

\( M = \{M₁, M₂, \ldots Mₖ\} \) is the set of mappings from process \( P₁ \in P₂ \) to \( P₀ \) respectively.

The collective behavior of a composition setting \( CS = (P₁, P₂, M) \) can be seen as product automaton \( P₃ = (S₃, E₃, T₃, s₃^0, F₃) \) of the participating automata where

\[ S₃ = (S₁ × S₂ × \ldots × Sₖ), E₃ = \cup_{i=1}^{k} E₁, S₃^0 = \{s₁^0, s₂^0, \ldots, sₖ^0\} \text{ and } F₃ = \{∀ i \in [1, k], (s₁ \in F₁ \lor s₂ \in F₂) \} \]

3.4 Relations

We introduce a set of relations to define the proposed integration properties, namely compatibility and completeness property. The relations are mainly for checking the process activities and the execution orders of the process activities of the involved process views.

3.4.1 Event inclusion relation

Given a mapping \( M \), two process automata \( P₁ = (S₁, E₁, T₁, s₁^0, F₁) \) and \( P₂ = (S₂, E₂, T₂, s₂^0, F₂) \), an event inclusion relation between \( P₁ \) and \( P₂ \), denoted by \( P₁ <ₘ P₂ \), is defined as

\[ \forall e₁ \in E₁, \exists e₂ \in E₂ \text{ such that } (e₁ \rightarrow e₂) \land \text{Type}(e₁) = \text{Type}(e₂) \]

Essentially, \( P₁ <ₘ P₂ \) states that for each event of \( P₁ \) there must exist a corresponding event of the same type in \( P₂ \).
3.4.2 Event Completeness

The event completeness criterion holds for a composition setting CS = (P_D, P_C, M) iff it satisfies the following condition:
\[ \forall e \in E_D, \exists e' \in E_C \text{ such that } (e = f_{\text{map}}(e') \land \text{Type}(e) = \text{Type}(e')) \]

Essentially, the event completeness states that for each event of P_D there must exist a corresponding event of the same type in P_C.

3.4.3 Simulation relation

Given a mapping M, two process automata P_1 = (S_1, E_1, T_1, s_1^0, F_1) and P_2 = (S_2, E_2, T_2, s_2^0, F_2), a relation R \in S_1 \times S_2 is called a simulation relation iff it satisfies the following condition:
\[ \forall s_1 \in S_1, s_1 \xrightarrow{e} s_1' \text{ and } s_2 \in S_2 \]
\[ \text{if } (s_1, s_2) \in R, s_1 \xrightarrow{e} s_1' \text{ then } \exists s_2' \in S_2 \text{ such that } s_2 \xrightarrow{f_{\text{map}}(e)} s_2' \cdot (s_1', s_2') \in R \]

Given a mapping M, a process automaton P_2 simulates process automaton P_1, denoted by P_1 \leq_M P_2, if there exists a simulation relation R \in S_1 \times S_2 such that (s_1^0, s_2^0) \in R. Essentially, for each transition path of P_1 starting from its initial state there exists a corresponding transition path in P_2 starting from its initial state.

3.4.4 Collective simulation relation

Let (P_D, P_C, M) is a composition setting where P_D = (S_D, E_D, T_D, s_D^0, F_D) is desired process automaton, P_C = (S_C, E_C, T_C, s_C^0, F_C) is collective process automaton of process automata P_1, P_2, ..., P_k and M is set of event mapping. A relation R \in S_D \times S_C is called a collective simulation iff it obeys the following conditions:
\[ \forall s_d \in S_D, s_d \xrightarrow{e} s_d' \text{ and } s = \{s_1, s_2, ..., s_k \} \in S_C \]
\[ \text{if } (s_d, s) \in R, s_d \xrightarrow{e} s_d' \text{ then } \exists s' \in S_C \text{ s.t. } s \xrightarrow{f_{\text{map}}(e)} s' \cdot (s_d', s') \in R \]

Where transition over the set of state of participating processes automata is as follows:
\[ s \xrightarrow{f_{\text{map}}(e)} s', \text{there exists at least a } i \text{ s.t., } s_i \xrightarrow{f_{\text{map}}(e)} s_i' \text{ where } i \in [1, k] \]

A given a set of mapping M, product of participating processes P_C simulates the desired process P_D, denoted by P_D \leq_M P_C, if there exists a simulation relation R \in S_D \times S_C such that there exists an s^0 \in s_i^0 that satisfies (s_D^0, s_C^0) \in R. Essentially, for each transition path of P_D starting from its initial state there exists at least one corresponding transition path in P_1, ..., P_k starting from their initial states.

3.5 Properties of interests

Compatibility property

A compatibility property is proposed to determine the extent of reusability of a process view with respect to desired process view and the adaptation required in case of a mismatch.

Given a mapping M, a process automaton P_1 is compatible with respect to process automaton P_2 iff following criterion is satisfied
\[ (\text{Restriction (P_2, P_1, M)} \leq_M P_1) \land (\text{Restriction (P_2, P_1, M)} \leq_M P_1) \]

We say that an existing process automaton P_2 is compatible with respect to the desired process automaton P_1 when events of P_1 include the events of Restriction (P_2, P_1, M) and P_2 simulates Restriction (P_2, P_1, M). This essentially states that restricted to the event set of P_1, there are no transition paths in P_2 that are not possible in P_1. If this was not the case, the desired process may take an existing process into an unanticipated state possibly violating its compatibility. The proposed approach checks the compatibility property and also in case of a mismatch, the verification generates a counter-example. The counter-example is a path from initial state to the state violating the compatibility property. One can mitigate (if possible) the mismatch by analyzing the counter-example.
Completeness Property

Completeness property ascertains that the desired process view can be realized by integrating the participating process views. A given composition setting \((P_D, P_C, M)\) is complete iff the following conditions are satisfied:

i) Composition setting is compatible i.e. \(\forall i \in [1,k] \), where \(k = |P_C|\), the compatibility criteria holds for process automaton \(P_i\) with respect to the desired process \(P_D\) i.e. \(\text{Restriction}(P_D, P_i, M_i) \leq M_i P_i\)

ii) Composition setting satisfies the event completeness relation.

iii) Desired process automaton can be collectively simulated by the set of participating process automata i.e. \(P_D \leq_{M_c} P_C\)

The verification process checks the completeness property and generates counter-example in case of any gap, mismatch or conflict. The rules for generation of counter-example are as follows:

i) If the participating process automaton of a composition setting \((P_D, P_C, M)\) does not satisfy the compatibility criterion with respect to the desired process automaton then there exists a mismatch between a participating process automaton and the desired process automaton. It generates a counter-example, a path from initial state to the state violating the compatibility property, to show the mismatch.

ii) If a composition setting \((P_D, P_C, M)\) does not satisfy the event completeness condition then there exists a gap between the desired process automaton and set of participating automata.

iii) During the analysis of collective simulation, if more than one participating process automata participates for a single transition of the desired process automaton then there exists a conflict among the participating process automata. We identify this conflict by analyzing the event sets of the participating process automata and event set of desired process automaton. Although this does not violate the completeness criterion but verification generates a counter-example in the form of name-value pair to represent the conflicts as information.

One can add more participating applications into the composition setting to fill the gap. Conflicts and mismatches can be mitigated by applying process mediation operators, if possible, or the application causing conflicts or mismatches can be discarded from the composition setting.

3.6 Process Mediation

The proposed approach provides a set of techniques for analyzing the compatibility property and completeness property. It also generates a counter-example in case of mismatches or conflicts in execution orders of the involved activities between the existing and the desired process views. A certain class of conflicts can be resolved by transforming the process view in mismatch or conflict into the desired process view using abstraction, refinement, hiding techniques. For instance, a process activity of a specific process view can be realized as a sequence of process activities in desired process view (refinement), a sequence of process activities of a process view can be considered as a single process activity in desired process view (abstraction), or an unintended process activity (like notification of a confirmation messages) is present in a process view, which is not required in desired process view. We term such mismatch as resolvable mismatch. The other class of mismatch can not be resolved by transformation techniques, we term this class of mismatch as irresolvable mismatch. For instance, if the process activities violate the execution order, i.e. process activity \(a2\) is executing after the execution of process activity \(a1\) in a process view, and desired process view expects \(a2\) should be executed prior to the execution of \(a1\).

We propose a set of process view mediation operators to mitigate the resolvable mismatch. The operators are as follows: Hide, Replace and Use. The Hide operator hides specified activity from a process view; this is useful for ignoring undesired notification kinds of events. The Replace operator replaces a process activity or a set of process activities (source activity) by another process activity or set of process activities (target activity). The replace operator fundamentally addresses the refinement and abstraction process. The Hide and Replace operators can be used to mitigate the mismatches detected during analysis of compatibility property. The Use operator is used in the context of composition setting. It mitigates the conflicted composition setting by replacing an activity of a participating process view by another activity of some different participating process view.
3.7 Tool

In order to automate the proposed formal techniques, we have implemented a toolset using fc2tools [16] and MONA package [17]. The toolset implements the primitive relations, namely *event inclusion*, *simulation* relations. The implementation assumes vocabularies of involved process automata are normalized into a uniform vocabulary. In particular, the mapping function returns the same event i.e. $f_{map}(e) = e$ hold for all events. Event Inclusion ($EventInclusion(P_1, P_2)$) implements the relation $Restriction(P_2, P_1) \preceq P_1$ and the Process view simulation ($ProcessViewSimulation(P_1, P_2)$) implements $Restriction(P_2, P_1) \leq P_1$ relation for given two process automata. The restriction operator is implemented using an algorithm similar to [18]. Since process automaton is a deterministic finite state automaton, we use a standard language containment algorithm to verify simulation relation. The language containment relation is implemented using MONA based DFA library functions. On failure of verifying relation, a counter-example is generated by querying (using dfaMakeExample) the DFA object.

We term this verification toolset as *process automata verification environment* (PAVE). PAVE uses an extended FC2 [16] format as the input specification language. The transition relations are specified using FC2 format, and it is extended with name-value pairs to specify events and their types. The toolset of the PAVE provides support for creation of DFA object from FC2 format and DFA object to FC2 format. The translation from DFA object to FC2 format is required for notification of feedback in case of failure. In present version of the PAVE does not support collective simulation relation. We compute the collective simulation relation by computing the product automata of existing processes and then checking the simulation relation. PAVE provide a native implementation of computing product automata and verifying the simulation relation on product automata.

4. Implementation of Process View Integration Framework

A formal approach for analyzing standalone process model in the context of EAI to improve upon correctness and precision of reusability, adaptation and integration is presented in previous section. The proposed approach is automated through a toolset, PAVE, implemented using Esterel, fc2tools and Mona package. The implemented toolset is based on well known automata theory. However, there is a large conceptual gap between the technical space prevalent in EAI domain and the technical space of proposed formal foundation. For instance, the gap exists in i) specification: the process view verification environment uses FC2 format as the input specification, which is neither prevalent in EAI practice nor has a direct mapping from the specification that are prevalent in EAI practice, ii) expressiveness in specification: typically the expressiveness of specific formalism is restrictive as compared to the expressiveness of the specification prevalent in EAI domain, for instance process automata can capture only the behavioral aspect of the enterprise application, and iii) requirement: typical formal toolsets address a part of the complete problem with limitations, for example PAVE assumes a uniform vocabulary for the involved process automata which is too restrictive for the practical scenario.

We propose a framework to bridge the abstraction gap between these two technical spaces wherein i) the various aspects of application model can be specified using modeling notations prevalent in industry, ii) the desired integration properties can be specified using high-level domain-specific languages and iii) the specified integration properties can be verified using a suitable formal and informal toolset. The framework is designed for extension by plugging in support for the relevant additional integration properties. The Model Driven Architecture (MDA) plays an important role, as it provides an approach and a framework for establishing bridges among various technical spaces, by providing domain integration and interoperability via metamodel mechanisms. However, if there is a large conceptual gap between the two technical spaces, defining a suitable MDA transformation is a highly non-trivial task. This work deals with the scenario in which the metamodel of the technical space of the source, i.e. EAI domain, is different from the metamodel of the technical space of the destination, i.e. formal foundations. This chapter presents an approach for bridging these two technical spaces using a framework.

The proposed framework is characterized by: i) Separation of concerns, ii) Modeling language most suitable for specifying the desired concern, iii) Specification of integration properties, iv) Support for model transformation. The key concepts and various characteristics are explained in below sub-section.
4.1 Models

An integration property can be composed of a set of primitive properties, \emph{primitive integration properties}. Typically these primitive integration properties can be characterized as the verification of one of the following: formal relations, algorithmic routines, or invariant over set of models or model elements, where the model-element can be a subset/projection of a complete model (with/without abstraction, refinement or specific transformation). In order to enable this heterogeneity in verification process, the framework supports three kinds of models as follows:

i) Application model: A high-level domain specific model to represent an application as a whole.

ii) Concern-specific model: This model captures a specific concern of interest of an application model. For instance, data model, services model and process view are the concern specific models of an enterprise application.

iii) Analysis-specific model: The framework supports analysis-specific model to use formal verification techniques as part of the complete analysis process. Typically a formal tool expects a specific formalism to verify a specific property of interest. In general, the expressiveness of these formalisms is restrictive as compared to the modeling notations prevalent in industry practice. A specific formalism captures a specific aspect of an enterprise application, for instance, finite state automata model is well-suited for representing the behavioral aspect of a process model, Pi-calculus is well-suited for representing the interaction aspect, etc. Moreover, these formalisms are too abstract a level to specify adequate information of a specific aspect of an application model. Analysis-specific model serves the purpose of interoperability between a specific formalism and concern specific model. In general, analysis-specific model contains two parts: structural part and behavioral part. The structural part is conforming to a specific metamodel with a place-holder for behavioral part and behavioral part is a textual specification conforming to a specific formal language.

The supported model hierarchy is based on MDA 3-layered modeling architecture concepts as shown in Fig. 2.

4.2 Domain-Specific language

Expressing the integration properties for various concerns of interests using a high-level language is one of key features of the framework. The properties of interest can be a combination of formal and informal verification, and thus it may work either on a concern-specific model or on an analysis-specific model. For instance, \emph{event completeness} relation is typically a structural verification over a set of process activities which can be specified using OCL [20]. On the other hand, proposed relations are, typically, based on formal techniques, like simulation, restriction, etc. In order to facilitate a uniform environment at higher-level of abstraction, a specification language, \emph{property specification language}, is proposed.
4.3 Framework Components

An overview of the framework, with various building-blocks, is depicted in the Fig. 3. At the heart of the framework is an integration bus (view integration bus) which acts as a mediator among the various framework components each serving a specific purpose. Additional components addressing other concerns of interest, such as data verification, service discovery etc., can also be plugged-in to the integration bus as per the needs of the specific EAI problem. The key building-blocks of the framework are described below:

**View Integration Bus**

The principle objective of the view integration bus is to provide an extensible integration environment wherein various tools (modeling tools, specification editors, transformation tools, and multiple verification environments) can be plugged-in.

**Property Specification Language**

The properties specification language (PSL) is designed to specify the integration properties at concern-specific model level and it can work either on concern-specific model or on a set of analysis-specific models. The integration properties are typically composed of several primitive integration properties specified over a set of model elements of concerns-specific model. The expression language of the PSL is based on OCL [20] and the structural constructs are based on first-order logic and predicate logic. The expression constructs allow model navigation and model manipulations such as transformation, filteration and instantiation. The structural constructs such as AND, OR, NOT, THERE EXISTS and FORALL facilitate definition of complex structure over a set of primitive integration properties. Ability to compose these constructs recursively provides flexibility and enables reusability at specification level. The language supports following primitive data-types: Boolean, Set, String and MAP, and minimal fault handling capabilities using ONERROR construct. In case of a failure in verification, a counter-example is generated, which is accessible to the user as pre-defined FEEDBACK object within ONERROR construct. Apart from these static features, the proposed property specification language also provides an extension point that can be used to extend the language by defining new primitive integration properties and the necessary operators.

**MDA Modeler**

MDA modeler provides support for defining various metamodels (conforming to a meta meta model) and models (conforming to a specific metamodel).
**Model Transformation Environment**

This component provides an environment to specify the transformation rules, at the metamodel-level, to automate the transformation process [21,22]. The transformation rules also help in providing traceability across the various modeling layers. The verification may generate a feedback (in case of any failure) that needs to be transformed back to the concern-specific model and then to the application model (if required).

**Formal Toolset Integrator**

The formal verification of primitive integration properties involves verification of formal relations or execution of algorithmic routines on a set of analysis-specific model. The formal relations and algorithmic routine can be verified using a specific formalism and an associated formal toolset. We term this setting (a formal toolset with a specific formalism) as a *formal environment*. Typically, these formalisms are not conforming to MDA standards. Moreover, a set of formal environments may require to verify a specific integration property (as an integration property is composed of several primitive integration properties).

Formal toolset integrator works as an adopter between *view integration bus* and a set of *formal environments*. The infrastructure of the formal toolset integrator is depicted in Fig. 4 (a). It consists of two main modules: an adaptor and property definer. The core of the formal toolset integrator can be seen as an *adaptor* with multiple ports. The source-end of the adaptor connects to the *view integration* bus and target-ends connect to various *formal environments*. Adaptor converts the concern-specific model into appropriate analysis-specific model by instantiating the appropriate model transformer and decides the correct target port for analyzing a *primitive integration property*. In case of failure, the adaptor instantiates appropriate model transformer to transform the feedback into concern-specific model in order to propagate back the feedback into the integration bus. Each of the connected formal environments provides a set of *primitive integration properties* as abstract interfaces to the framework. The proposed framework uses these interfaces to extend the property specification language. Property definer module exports the supported primitive integration properties in a precise format wherein the parameters of the primitive integration properties are the model elements of *concern-specific metamodel*. The syntax of the proposed format is described in Fig. 4 (b).

**Model Based Verification Environment**

Model Based Verification Environment (MBVE) is an environment for analyzing the MDA-compliant model elements. Typically, it verifies the invariants over a set of model elements with or without transformation. Framework uses OCL and its toolset to realize model based verification environment. The environment provides support for exporting the supported operators and primitive integration properties to the framework as an abstract interface using a module similar to property definer module.

---

*Fig. 4: (a) Formal toolset integrator (b) property definer*
**Analysis Engine**

Analysis engine serves the purpose of a virtual machine of the property specification language. This provides a common environment to work on a high-level specification language that is transparent to the underlying formal toolsets being used. The instruction set of property specification language can be divided into three subsets: i) instructions for model element manipulation i.e. instantiations, accesses - this type of instruction is executed on concern-specific model by using MBVE, ii) invocation of mediation operator - these are mainly model transformation rules, executed on concern-specific model, used mainly for process mediation or process view adaptation, and iii) invocation of primitive integration properties offered by the set of formal environment. Analysis engine propagates these instructions to the formal toolset integrator. Formal toolset integrator propagates them to the appropriate formal environment by transforming the appropriate analysis-specific model and relaying the feedback after transforming the counter-example into the concern-specific model.

**4.4 Extendibility and Separation-of-concerns**

By adopting the basic principle of MDA approach and MOF [23] three-layers model architecture as the core of the proposed framework, we facilitate the extendibility and separation of concerns to maximum possible extent. The framework can be extended in many directions. One can plug-in additional framework components that address other concerns of interest, like, validation of data model, service model etc. Other formal environments, addressing the interaction aspect or other formal aspects associated with data, service and process models, can also be plugged-in to the framework.

As a whole, we facilitate the separation of concerns in various dimensions. In one direction, the framework provides support for separating the various concerns of interest of an enterprise application. This provides an environment to use specialized approach for each of the concerns. On the other direction, it provides support for separating the complex verification problem into a set of aspects by transforming a concern-specific model into a set of analysis-specific models. Application of specific formal tools and techniques, based on different formalisms, for each of these aspects is also possible in the framework. Even the verification of specific integration property can be a combination of formal and informal techniques, wherein the informal techniques can be evaluated on concern-specific model and formal techniques can be evaluated on appropriate analysis-specific model. The support for high-level specification language for specifying application model and a general purpose property specification language hides these heterogeneous verification mechanisms, based on various tools and models, from the end-users.

With the multiple of concerns specific models and separate treatment for each of these concern specific model, being able to establish the relation between various concern specific model-elements should be a feature of the proposed framework. This calls for a model weaver to weave various feedbacks into a consistent whole. As we are dealing with a process model in the proposed framework, the model weaver building-block is not considered at this stage.

**4.5 Implementation Details**

In earlier sections we have presented a generic framework to analyze various aspects of the enterprise application. It supports a high-level modeling language to specify the enterprise application as a whole, various models to represent different concerns and aspects of an enterprise application, and a property specification language to specify the desired integration properties at a higher-level of abstraction. In this section we present an implementation details to customize the proposed generic framework to support the process view integration. We term this as process view integration framework (PVI framework). We use UML profile for business process modeling [4] to specify enterprise application and PAVE as a formal environment to verify the primitive integration properties in PVI framework. As part of the realization of PVI framework, we define i) process view metamodel - a specific metamodel for representing the behavioral aspect of the process model of an enterprise application, ii) Analysis-specific metamodel - a metamodel for analysis-specific model, iii) transformation strategy and rules - transformation rules to transform process view to process automata, and vice-versa (for feedback), iv)
process view mediation operators and primitive integration properties, and v) integration properties using proposed PSL.

4.5.1 Process view metamodel

Process view describes the behavioral aspect of an enterprise application with the built-in assumptions of the involved process activities in the implementation made explicit. A process activity, an atomic unit, is either a service offered/required by an enterprise application or a manual task. The abstract specification of the process view can be described using various control flow constructs over a set of process activities. BPEL4WS supports various control flow constructs to specify a business process. This provides support to specify different aspects, i.e. behavioral aspect, integration aspect and execution aspect, of a process model. We propose a metamodel aligned with BPEL4WS specification to specify the behavioral aspect of an enterprise application. The proposed metamodel is described in Fig. 5. Process, Activity, Composition Settings are the main model-elements of the proposed metamodel. A Process is composed of various Activities. An Activity can be of two kinds: basic activity and structured activity. The semantics of all supported nodes are similar to BPEL4WS constructs. A Guarded Activity is either a basic or a structured activity with a guard condition. A guard is an expression of any of the following type: Boolean expression, time expression, or specific event expression. A Composition setting has a desired process and a set of participating processes. In order to capture the feedback of the analysis, a Feedback model-element is added with the process view model.

Fig. 5: Process view Metamodel
4.5.2 Analysis-specific metamodel

We use process automata as the formal model to describe a process view. A process automaton is typically a set of events (vocabulary) and their transition relations. In order to achieve the interoperability between process view model and process automata, the concept of analysis specific model is proposed in our framework. A *process automata specific* metamodel is proposed to describe the analysis-specific model for process automata. The metamodel has two parts: structural and behavioral. The structural part describes the set of events and their types. The behavioral part describes the transition relation using a textual format. We use FC2 format to describe the relations. The proposed metamodel is depicted in Fig. 6. A *feedback* model-element is added in the proposed metamodel to describe the generated feedback.

![Fig. 6: Analysis-specific Metamodel](image)

**Fig. 6: Analysis-specific Metamodel**

**Fig. 7: Mapping strategy between concern-specific model and analysis-specific model**

![Fig. 7: Mapping strategy between concern-specific model and analysis-specific model](image)
4.5.3 Model transformation

We use a high-level model, application model, to describe an enterprise application. Three kinds of models, namely application model, concern-specific model and analysis-specific model, are used to represent various concerns of interest and specific aspects of each of these concerns. We use UML profile for business process to specify an enterprise application. Two metamodels, namely process view metamodel and process automata specific metamodel, are proposed to describe process view and process automata respectively. We use BPEL4WS to represent the feedback in a textual format.

In order to establish the interoperability between the supported models, we define a transformation strategy and a set of transformation rules. We extract the process view model (conforms to process view metamodel) from an application model (described using UML Profile for business process) to analyze the process model separately; and then we translate (if required) process view model to process automata specific model to verify formal primitive integration properties using PAVE. The verification may generate the feedback (in case of a failure) that needs to be transformed back to the process view model and then to BPEL4WS specification for complete traceability. Required transformation rules are:

1. Transformation between application model and process view model (UML2ProcessView): the process view metamodel is aligned with BPEL4WS constructs. The transformations from UML Profile for business process to BPEL4WS and vice versa are defined in [4]. We use the same strategy for this transformation. We use ETTK toolset [25] to automate this transformation.

2. Transformation between process view model and process automata specific model (ProcessView2ProcessAutomata): We define a bi-directional mapping strategy between process view metamodel and structural part of the process automata specific metamodel. The proposed schema is conforming to QVT model-to-model transformation strategy [21]. The overview of the proposed schema is depicted in Fig. 7. This scheme automates the transformation of process view model to structural part of the process automata specific model and transformation from feedback to process view model.

3. Transformation from process view model to process automata specification (ProcessView2FC2): The complete translation of process view model the process automata specification (FC2) is a two-step process. First we translate the process view into Esterel program [26] using QVT model-to-text transformation schema [22] and then we use Esterel compiler and FC2Tools to compile the Esterel program into FC2 format. Aim of this transformation is to generate the correct FC2 specification (process automata transitions) but not the generation of equivalent Esterel program. We have chosen pure Esterel [26] as an intermediate language for the following reasons: i) it is an event based textual language that supports constructs for parallelism, choice, looping, termination, and the desired error handling, ii) Esterel environment provides support for generation of finite state deterministic automata by using Esterel compiler and FC2Tools, and iii) Suitable transformation rules for transforming a process view to Esterel specification can generate the desired automata for all workflow patterns [24] that are supported by BPEL4WS. Also, Esterel event relation provides an adequate support to describe the dependency between involved events. For example, the parallelism with link construct, the modeling challenge 4 in [24], can be described using parallel construct and event relation (the transformation rule for flow with link in table 1). By considering all process activities, offered (reply) as well as required (invoke and receive), as a set of input events of the Esterel program and providing appropriate rules to generate correct automata, we get over the mismatch of asynchronous and synchronous paradigm. The set of rules for transforming process view into Esterel is described in appendix.

4.5.4 Operators and primitive integration properties

The proposed approach provides a set of techniques for analyzing a set of integration properties, namely, compatibility property and completeness property. Typically, each of these integration properties is a composition of a set of primitive integration properties. For instance, compatibility property comprises of event inclusion relation and process view simulation relation. Each of the primitive integration properties can be verified either by using a model based toolset or by using a formal toolset based on the nature of verification. The framework uses model based verification environment and a set of formal environments to verify these two classes of verifications respectively. In order to address the process view integration problem, we use process automata verification environment (PAVE) as a formal environment. PAVE
supports three primitive integration properties, namely, event inclusion, process view simulation and collective simulation, which are exported using property definer module as EventInclusion(\text{Process, Process}), ProcessViewSimulation(\text{Process, Process}), and CollectiveSimulation(\text{CompositionSetting}). The model based verification environment provides event completeness relation as the primitive integration property, which it can verify. The supported integration property is exported as follows: EventCompleteness(\text{CompositionSetting});

A primitive integration property generates a counter-example in case of a failure. Essentially this failure is due to the conflicts in execution orders of the involved activities between the existing and the desired process views. We proposed a set of process view mediation operators to mitigate identified conflicts or mismatches. Ideally a process view which is in conflict can be mitigated to fit into the context of desired process view using view transformation technique. The application of process view mediation operator depends on the business context and the business semantics. For example, whether a process activity, typically the activity of type \textit{reply}, can be ignored or not depends on the business user, similarly whether a set of process activities (sequence activity) can be replaced by a single activity (basic activity) also depends on the business semantics of each of the involved process activities. Hence the application of these operators entirely depends on the business users, which they can decide based on the counter-examples. The supported process view operators are exported as follows: \textit{Hide(\text{Process})}, \textit{Replace(\text{ProcessActivity, ProcessActivity})}, and \textit{Use(\text{ConfigurationSetting, ProcessActivity, ProcessActivity})}.

The transformation technique is based on the proposed process view metamodel. \textit{Hide} operator can hide a basic or structured process activity. \textit{Replace} operator can be used for replacing a basic process activity by structured process activity and vice-versa. The \textit{use} operator can be used for the same type of BPEL4WS activity within a composition setting. Two constructors, namely \textit{ProcessView(\text{Application Model})} and \textit{CompositionSetting()}, are proposed to construct process view and composition setting. \textit{ProcessView} constructor constructs a process view from the application model. This constructor encapsulates the \textit{UML2ProcessView} transformation process. \textit{CompositionSetting} constructs an empty composition setting. The desired process view and participating process views can be attached with the composition setting using property specification language constructs. As PAVE expects the normalized vocabulary of the existing or the participation process views with respect to the desired process views, we
apply the mapping over the process activities on process view model. In order to achieve this uniform vocabulary in two analysis specific views, we propose an operator Mapping (MAP) to apply the mapping before verification.

5.5.5 Property specification

In this implementation, the property specification language is extended by importing the primitive integration properties offered by PAVE, MBVE and the proposed process view mediation operators. This raises the abstraction to process view model level by ignoring the underlying toolsets. Proposed compatibility property and completeness property are specified using the extended property specification language. The sample specifications for compatibility property and completeness for process views are described in appendix section (fig. A1).

4.5.6 Overview of the analysis process

Required models, metamodel, transformation rules, verification environment and process mediation operators are presented to realize the proposed framework. This section provides an overview of the analysis process to verify the integration properties of the process views. The analysis process proceeds as follows. The specification of application models and an integration property which user would like to verify on process view need to be specified using UML profile for business process and property specification language respectively. The proposed framework checks the desired property and provides feedback. In case of any mismatch, the one can apply process mediation operator and recheck the property on the mediated process views.

The verification process of the proposed integration properties using the process view integration framework is depicted in Fig. 8. The process view model, conforming to process view metamodel, is extracted from the application model. The process view model is translated into process automata specific model. The desired integration property is typically a composition of primitive integration properties. Each of these primitive integration properties is verified by executing on the analysis engine. Analysis engine uses model based verification environment for model manipulation and structural verification. It uses formal toolset integrator for reasoning the formal verification. The formal toolset integrator instantiates appropriate translator to translate the process view into desired formalism and selects corresponding toolset. In particular, it uses ProcessView2ProcessAutomata to transform the structural part of the process automata model and ProcessView2FC2 for transforming the process view into FC2 format. Verification mechanism generates counter-example in case of any mismatch. Formal toolset integrator converts that feedback into a process view specific model. The bidirectional ProcessView2ProcessAutomata translates this to process view model to show the mismatch to the end-user.

4.6 Example

In this section, we evaluate our approach and the framework using a scenario from travel industry domain. Typically, the requirement in travel industry is to build integrated applications that comprises of various services from different service providers, like Airlines companies, car rental companies, hotel agencies, etc. Traditionally customers have had two possible alternatives in traditional travel reservation process to reserve the required number of seats, cars, hotel rooms or berths for the required duration by directly approaching the service provider or to use the services of travel agents.
With the growing acceptance of Internet, electronic businesses, and popularization of merger and acquisition processes, the travel industries are gradually changing their business models. As the Internet and web based systems are good at providing an alternative channel bypassing various intermediaries for

Fig. 9: Process automata of CRS of airlines Companies
(a) Process automata of CRS of Airlines Company A, (b) Process automata of CRS of Airlines Company B

Fig. 10: (a) Process Automaton of CRS of Hotel H, (b) Process automaton of a local transportation service.
selling the inventory of service providers, the major service providers are trying to provide one-stop-shop facility to its potential. In most of the cases, these service providers themselves are trying to carry out the traditional role of travel agencies by offering bundle of options to its customers, wherein they can not only directly sell their inventory to the customer but will also seamlessly cross-sell inventory of their alliance partner, e.g. an airline company can provide a package offer inclusive of accommodation, ground transportation and other services such as travel insurance, foreign exchange, local tours for its passengers.

Such integrated service involves multiple service providers and requires uniform integration of their Central Reservation Systems (CRSs). Typically, these CRSs are being developed by various service providers for a specific purpose. Maximal reuse of these applications to provide value added services to its customer is the main challenge in this evolution. Our approach can be use in this scenario to identify the correct applications (CRSs) for a desired integrated application and analyze the conflicts and gaps between existing CRSs of involved service providers and desired CRS of integrated service.

As an example, we consider CRS of two Airlines Companies (Company A and Company B) and a CRS of a hotel company (Hotel H). Suppose Company A and B facilitate the air ticket reservation and cancellation process to their customer. Wherein Company A accepts credit card and cash payment and Company B accepts credit card and pay-order for reserving tickets. Hotel H facilitates a process to reserve their rooms and support for various tariffs related queries. The process views of these existing applications, in the form of finite state automata, are depicted in the figures 9 (a), 9 (b) and 10 (a). Figure 9 (a) and (b) present process view of CRS of Company A ($P_{CRS,A}$) and Company B ($P_{CRS,B}$) respectively. The process view of CRS of Hotel H ($P_{CRS,H}$) is depicted in figure 10 (a). Suppose an alliance of Companies A, B and hotel H wants to provide a new service to their customers wherein the customers who reserve their tickets with Company A or B can avail a special discounted rate while booking transit accommodation in Hotel H and also can avail local transportation services. The desired process view of the alliance companies, $P_{desired}$, is depicted in the Figure 11.

In first phase (discovering correct applications for a desired application), the Alliance Companies would like to analyze their existing applications for reusing them into the desired context. In case of any mismatch, the company would like to know if the existing applications could be made to fit with some adaptation. In the second phase (composition definition), the fitting applications, with or without adaptation, can be considered for integration in order to realize the desired application with assurances of
completeness of the integration. The compatibility and completeness properties can be used to analyze the behavioral aspect (process views) of the existing applications with respect to the process view of desired application. The process views of existing applications with respect to the process view of desired application can be analyzed using proposed compatibility criteria and the behavioral completeness of a set of existing applications with respect to a desired application can be analyzed using completeness criteria by constructing a composition setting. The analyses also determine the mismatch, conflicts and existing gaps. For example, given a mapping $M_1 = \{\text{Display Available Service} \rightarrow \text{Display Menu List}\}$, the process level compatibility property is satisfied for $P_{\text{CSR_A}}$ with respect to $P_{\text{desired}}$. In this scenario, the source process automaton is compatible with respect to target process automaton i.e. $\text{Restriction}(P_{\text{desired}}, P_{\text{CSR_A}}, M_1) \leq P_{\text{SRC_A}}$. This implies that the execution orders of all involved activities (traces) in the desired process automaton are compatible with the existing process automaton. Hence the $P_{\text{SRC_A}}$ can be used in the context of $P_{\text{desired}}$. In contrast, given a mapping $M_2 = \{\\}$, the process automaton $P_{\text{CRS_B}}$ does not satisfy the compatibility property with respect to $P_{\text{desired}}$, i.e. $\text{Restriction}(P_{\text{desired}}, P_{\text{CRS_B}}, M_2) \not\leq P_{\text{CRS_B}}$. If we analyze the restricted process automaton of $P_{\text{desired}}$ described in figure 12, there exists a path where a ticket can be issued without any payment. The desired process view accepts two types of payment modes, credit card payment and cash payment, but $P_{\text{CRS_B}}$ accepts pay-order instead of cash payment. The existing application $P_{\text{desired}}$ cannot handle the pay-order processing, and hence the mismatch. Another mismatch in $P_{\text{CRS_B}}$ process view is the order of execution of activities $\text{Cancel Ticket}$ and $\text{Refund Payment}$ are not matching with the desired process view. The restricted process automaton with the mismatched states, as gray color states, is depicted in the figure 12. The verification of $P_{\text{CRS_H}}$ with respect the desired process view ($P_{\text{desired}}$) also fails to satisfy the compatibility property. Given a mapping $M_3 = \{\text{Accept Payment} \rightarrow \text{Accept Ticket Details}\}$, the existing process view $P_{\text{CRS_H}}$ sends a reminder to the requester about the booking which is not present in desired process view. So the restricted process automaton of $P_{\text{CRS_H}}$ fails to simulate the desired process automaton $P_{\text{desired}}$. The feedback is generated by showing the counter-example, i.e. a path from initial state $s1$ to a state $s7$ in process automaton of $P_{\text{CRS_H}}$, which is depicted in the figure 10(a). The existing process view is sending an unintended message (reminder notice), which is not desired in desired process view. This mismatch can be mitigated if it is possible to hide the unwanted notification. The

![Fig. 12: Restricted Process automaton of $P_{\text{CRS_B}}$](image)
alliance company can decide whether the notification can be ignored, without affecting any business value, or not. Suppose they decide the notification can be ignored. The verification condition will be satisfied if we hide the unwanted notification using Hide operator and perform the verification.

At the end of discovery phase, the alliance company can get a detail view about its existing applications, i.e., which applications are behaviorally compatible (with or without adaptation) for a desired application. In this example, process view \( P_{CRS_A} \) and adapted process view \( P_{CRS_H} \) are compatible with the desired process and hence can be selected as participating process views to realize a desired process view \( P_{desired} \). A composition setting (CS) can be constructed for this requirement as follows:

\[
P_{D} = P_{desired}, P_{C} = \{ P_{1} = P_{CRS_A}, P_{2} = P_{CRS_H} \} \text{ and } M = \{ M_{1}, \{ \} \}.
\]

This composition setting is compatible since compatibility criteria holds for all participating processes. The composition setting, CS, does not

---

**Fig. 13: The specification for checking compatibility properties**

INCLUDES
// include property specification file
@USE "property.psl" // property specifications are described in fig A.1

CONSTRUCTIONS // instantiation of process elements
Process \( P_{CRS_A}, P_{CRS_B}, P_{CRS_H}, P_{desired} \):
MAP \( M_{1}, M_{2}, M_{3} \):
\( P_{CRS_A} \) = ProcessView(ProfileA);
\( P_{CRS_B} \) = ProcessView(ProfileB);
\( P_{CRS_H} \) = ProcessView(ProfileH);
\( P_{desired} \) = ProcessView(ProfileD);
\( M_{1} \) = \{Display Available Service -> Display Menu List\};
\( M_{2} \) = \{Accept Payment -> Accept Ticket Details\};

// Verification 1: Verify the compatibility of CRS of company A
VERIFY {
  @CompatibilityProperty(\( P_{desired} \rightarrow P_{CRS_A} \rightarrow Mapping(M_{1}) \));
}
ONERROR {
  DISPLAY FEEDBACK;
}

// Verification 2: Verify the compatibility of CRS of company B
VERIFY {
  @CompatibilityProperty(\( P_{desired} \rightarrow P_{CRS_B} \rightarrow Mapping(M_{2}) \));
}
ONERROR {
  DISPLAY FEEDBACK;
}

// Verification 3: Verify the compatibility of CRS of hotel H
VERIFY {
  @CompatibilityProperty(\( P_{desired} \rightarrow P_{CRS_H} \rightarrow Mapping(M_{3}) \));
}
ONERROR {
  DISPLAY FEEDBACK;
}

// Verification 4: Verify the compatibility of CRS of hotel H
VERIFY {
  @CompatibilityProperty(\( P_{desired} \rightarrow P_{CRS_H} \rightarrow Hide(Reminder notice) \rightarrow Mapping(M_{3}) \));
}
ONERROR {
  DISPLAY FEEDBACK;
}
satisfy event completeness criteria. This implies that the desired process view $P_{desired}$ cannot completely be realized by using process view $P_{CRS,A}$ and adapted process view $P_{CRS,H}$. The verification generates feedback by showing the set of activities which are expected in the desired process view $P_{desired}$ but not presents in any of the participating process views. The set of activities are {Notify Cancellation, Pick-up Request, Display pick-up points, accept pick-up points, Get Ticket Details, Display Confirmation}. The alliance company can analyze the feedback in the context of desired process view and add more participating process views to the composition setting, CS. In this case the alliance company can add a service-provider for pick-up service. Suppose the Alliance Company acquire a local transportation company, whose processes is similar to the process view $P_{CRS,LT}$, which is depicted in figure 10. (b). If user verifies the completeness property by adding the new process view (P_{CRS,LT}) with the existing composition setting CS, verification will show a complete composition setting. Although there are some conflicts in the composition setting CS, such as the activity Display Menu List is present in multiple process views, but these can be mitigate by Use operator.

The complete analysis process can be automated through the proposed framework. The framework supports a UML profile for business process as the input specification of the participating and desired applications/ business processes. We suppose the business processes of Airlines Company A, Airlines Company B and Hotel H are described using UML profile for business process, and these are ProfileA,
ProfileB and ProfileH respectively. ProfileD is the UML profile for business process of the desired process. The specification for checking compatibility properties of various existing applications are described in Fig. 13. As we have seen that the process view \( P_{CRS_A} \) satisfies that compatibility criteria, the verification 1 of Figure 13 does not generate any feedback, whereas the verification 2 and 3 generate appropriate feedbacks showing the mismatches. These are as follows: i) feedback of verification 2 is [Display Menu List -> Reservation Request -> Seat Available -> Calculate Cost -> Issue Ticket ], ii) feedback of verification 3 is [Display Menu List -> Accommodation Request -> Accept Accommodation Details -> Accept Payment -> Notify Booking Details -> Send Reminder]. User can analyze these feedbacks and recheck the property with appropriate mediator operators. For instance, \( P_{CRS_H} \) can be adopted appropriately to make it compatible, which is depicted in verification 4 in Figure 13. Similarly the completeness property can be verified by constructing appropriate composition setting. In Figure 14, two composition settings \( cs_1 \) and \( cs_2 \) have been constructed, where composition setting \( cs_1 \) comprises of \( P_{CRS_A} \) and \( P_{CRS_H} \) and composition setting \( cs_2 \) comprises of \( P_{CRS_A} \), \( P_{CRS_H} \) and \( P_{CRS_LT} \) respectively. As we have seen earlier that \( cs_1 \) is incomplete and \( cs_2 \) is complete but there are some conflicts due to [Display Menu List activity. The feedbacks are generated as follows [ Incomplete => { Notify Cancellation, Pick-up Request, Display pick-up points, accept pick-up points, Get Ticket Details, Display Confirmation} ] and [ Conflict => { Display Menu List}].

5. Related work

Building a semantically correct system by assembling a set of existing systems is a well known paradigm in several engineering disciplines. For instance, the component based development [27], used widely in VLSI circuit design methodology, addresses a similar kind of problem using various formal tools and techniques. Formal techniques such as model checking, a variety of equivalence relations and their pre-order relations with a range of refinement techniques have been used with varying degrees of success to verify the properties of interest in VLSI design, embedded systems and various mission critical reactive systems. Recent times are witnessing attempts to apply these techniques to enterprise application integration. Model checking approaches have been used to check reachability of a process state [28]. Model checking and simulation techniques have been used for a semantic verification of UML activity graph [29]. Another approach for semantic verification of UML activity diagrams using finite state automata can be found in [30]. The approach presented in [13] checks language containment property between two business processes. The proposed approach uses finite state automata extended by logical expression associated with the process states to analyze compatibility of business processes. Howard Foster et al propose a technique to check the compatibility of business processes using trace equivalence relation, which is based on finite state process model [14]. Existing techniques for checking compatibility in the context of process reusability are based on model checking, system equivalence relation, or their pre-order relations. Model checking techniques are good for deadlock detection and state reachability problems but do not check the compatibility as a whole. The approaches based on equivalence relations, for instance trace equivalence, observational equivalence etc., are well suited for implementation verification problem, i.e. checking the correctness of an implementation with respect a specification. Checking equivalence of an existing process model with respect to the desired process model is too restrictive a relation to check compatibility. The concept of pre-order relations, such as simulation, trace inclusion, etc., are significant to address compatibility problem as whole. However, these relations need to be suitably adapted in order to address the practical issues such as different terminology in process activities of two process models, incompatible vocabularies, i.e. vocabulary of one process model need not to be a subset or a superset of the vocabulary of other process model. We use simulation technique with appropriate refinements, using mapping and restriction operators, to address this practical problem without compromising on rigor.

Several variants of Petri-Net and process calculi have been used to formalize business processes [5, 13, 14]. As we are dealing with the behavioral aspect of the stand-alone business process and the scalability of Petri-Net as a modeling technique for real world scenario is questionable in practice, we advocate FSM based model as the analysis specific model in our framework. On the other hand the analysis technique presented in [14] verifies the business process in the context of business process choreography. This considers the interaction aspect of the business process to determine how well a business process can communicate with the other business processes. Similarly the work presented in [13] also verifies the interaction aspect of the business process using language containment approach in the context of business
process choreography. In contrast, the proposed compatibility property analyzes the behavioral aspect of the stand-alone business process to determine the degree of reusability in the desired context. The Roman model and K\textsuperscript{th}–look-ahead delegator techniques have been proposed to address automated service composition [33,34]. They proposed a framework that describes the behavior of a web service as a finite state machine and an algorithm for checking the existence of a composition mediator. On the other hand, the approach presented in this paper analyzes the composition setting and generates a counter-example in case of incomplete composition setting so that one can adapt the composition setting by adding new application to fulfill the requirement of the desired application.

Area of semantic web addresses the issues of service compatibility and composition through ontology-based approached such as DAML-S [35]. In [36, 37], the issue of service composition is addressed in order to create composite service by reusing and extending the existing services using planning under uncertainty and constraint satisfaction techniques. Here the focus is on synthesizing an execution order of the involved activities that satisfy the client goals whereas our work focuses on obtaining the desired application process by reusing the existing application processes. In planning techniques, the devised composition can be exploited only once whereas our composition setting can be reused in multiple integration contexts that satisfy the safety property and completeness property. The existing approaches are either based on formal techniques, which are described above, or based on ad-hoc techniques supported by toolsets lacking a formal underpinning. This paper presents a pragmatic approach that combines high level notations used in industry practice for specifying processes of enterprise application with the rigorous analysis supported by formal techniques in the form of an automation tool that bridges these two worlds.

6. Conclusion

This paper focuses on the analysis of process views of enterprise applications in the context of reusability, adaptation and integration. It proposes a formal model suitable for rigorous behavioral analyses of the process view and a set of formal techniques to analyze various integration properties with increased precision. An approach that provides guarantees of correct application integration is presented along with the necessary tool support. The proposed compatibility property addresses the discovery of reusable processes and the proposed process mediation operators address adaptation of processes when required. The proposed completeness property determines if the desired process can be completely realized in terms of the existing processes as specified in the composition definition. As we have seen that the abstractions used in industry practice for representing enterprise applications/business processes, largely based on the ability to specify various concerns of interests of an enterprise application, are not amenable for analysis and verification. Amongst the standards for describing business processes, BPEL4WS and UML Profile for business process appeared the most completed notations and has been reported under consideration for both academic and industry projects. Our approach to modeling business processes has focused on the standards used in industry. We use the standard notations (in this case BPEL4WS/UML Profile for business process) to specify business processes and transform this specification into analysis-specific model (in this case process automata model) for rigorous analysis. Proposed process automata has ability to specify the behavioral aspect of enterprise application and also satisfies the requirements to being able to use for formal verifications using various formal toolset. The proposed framework provides necessary support for transformation to achieve the interoperability among the supported models. The transformation process abstracts the behavioral aspect of the high level specification and builds a model representing the behavior in the form of process automata. The framework, based on the basic principles of MDA approach and separation of concerns, is designed to bridge the gap between gap between the technical space prevalent in EAI domain and the technical space of formal foundation. The approach advocating in this framework is based on successive metamodel refinements and the refinement of integration properties to achieve seamless interoperability between the EAI technical space and the technical spaces of various formal foundations. The proposed framework provides an extendable environment to the domain experts and academicians to exploit various practices, methodologies and formalisms for providing more comprehensive solution. A comparative study can be performed using the proposed framework to determine which formalism and technique is best suited for enterprise application integration problem. A holistic approach combining data, service and process models to solve the enterprise application integration problem is left as the future work.
References

APPENDIX

A.1 Transformation Rules

Table 1 describes the transformation rules for each of the nodes of process view metamodel to transform process view model to the Esterel specification. The rules are described using a transformation language similar to OCL wherein "::" is similar to print statement, "$" is used to specify the model element, "." is used for model navigation and "@" is used for invoking other transformation rules.

Table A.1: Process view to Esterel transformation rules

<table>
<thead>
<tr>
<th>Process View Node</th>
<th>Process View model to Esterel Transformation rules</th>
<th>Description</th>
<th>Example (Esterel Code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process &lt;activities&gt;</td>
<td>.module $Process.Name :trap PROCESS_$Process.Name in @transform($Node.contains) :end trap :end module</td>
<td>Transformation of complete process.</td>
<td>module: ABC trap PROCESS_ABC in // Activities end trap end module</td>
</tr>
<tr>
<td>Terminate</td>
<td>:exit PROCESS_$Process.Name</td>
<td>Process termination</td>
<td>exit PROCESS_ABC</td>
</tr>
<tr>
<td>Sequence</td>
<td>//use : (sequence operator) Forall Node in $Node.contains : @transform($Node.contains) ;</td>
<td>Sequential flow of activities A1, A2, …, Ak</td>
<td>A1; A2; …; Ak.</td>
</tr>
<tr>
<td>Process Activity</td>
<td>nothing; case &lt;Otherwise.Identifier&gt; exit LOOP_K; end await; // Process Activity related code end loop;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case otherwise do nothing; case &lt;Otherwise.Identifier&gt; exit LOOP_K; end await; // Process Activity related code end loop;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow with link</td>
<td>Parallel execution of activities A1, A2, …, AK with explicit execution dependency. (Suppose Ai can execute only after execution of Aj)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional flow: Switch Case (Boolean-expression) Process Activity Case … otherwise Process Activity</td>
<td>// event declaration relation Ai =&gt; Aj</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>//body of the program [A1</td>
<td></td>
<td>A2 …</td>
</tr>
<tr>
<td>Pick</td>
<td>await case &lt;Operation.Name&gt; do … case &lt;Otherwise.Identifier&gt; do … end await;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Handling Node</td>
<td>await case &lt;Operation.Name&gt; do … case &lt;Otherwise.Identifier&gt; do … end await;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description: &lt;scope&gt; [activity with throw] &lt;&lt;&lt;Catch&gt;&gt; &lt;/scope&gt;</td>
<td>Support for error handling within a scoped activity: &lt;Scope&gt; catch // errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow //use</td>
<td></td>
<td>(parallel operator) Forall Activity in Node.contains : @transform($Activity)</td>
<td></td>
</tr>
<tr>
<td>Flow //use</td>
<td></td>
<td>(parallel operator) Forall Activity in Node.contains : @transform($Activity)</td>
<td></td>
</tr>
<tr>
<td>Conditional //Use Await ... Case... Do... end await : Await Forall Case in $Node.cases : Case &lt;Case.Expression.Identifier&gt; do @transform($Case.contains) If Exist($Node.Otherwise) : Case otherwise do @transform($Otherwise.contains) Else : Case otherwise do nothing; end await; @transform($Node.contains)</td>
<td>Conditional flow: Switch Case (Boolean-expression) Process Activity Case … otherwise Process Activity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.2 Property Specification

The specification for proposed compatibility properties and completeness properties using proposed property specification language are described in fig. A.1. The primitive integration properties are represented in bold.

```
//property specification file: IntegrationProperty.psl
PROPERTIES
    // P1 is desired process and P2 is existing process
    CompatibilityProperty (Process P1, Process P2)=>
        @EventInclusion($P2, $P1);
        @ProcessViewSimulation($P2,$P1); %%

    CompletenessProperty(CompositionSetting cs )=>
        FORALL process IN $cs. hasParticipatingProcess
        / @CompatibilityProperty($cs.desired, $process);
        / @EventCompleteness($cs);
        @CollectiveSimulation($cs);
    %%
```

Fig. A.1: Property specification of compatibility property and completeness property