

A Formalized, Taxonomy-Driven Approach to Cross-Layer Application Adaptation

RAZVAN POPESCU and ATHANASIOS STAIKOPOULOS, Trinity College Dublin
ANTONIO BROGI, University of Pisa
PENG LIU and SIOBHÁN CLARKE, Trinity College Dublin

Advances in pervasive technology have made it possible to consider large-scale application types that potentially span heterogeneous organizations, technologies, and device types. This class of application will have a multilayer architecture, where each layer is likely to use languages and technologies appropriate to its own concerns. An example application is a geographically large-scale crisis management system. Typically, such applications are required to dynamically adapt their behavior based on current circumstances, with adaptations potentially affecting all layers of the application. The complexities involved in dynamically adapting multilayer applications will significantly benefit from formal approaches to its specification.

This article presents a new methodology for flexible, multilayer application adaptation, with layer-specific adaptation solution templates bound to application mismatches that are organized into hierarchical taxonomies. Templates can be linked either through direct invocations or through adaptation events, supporting flexible cross-layer adaptation. The methodology illustrates the use of different formalisms for different elements of its specification. In particular, we combine semiformal metamodeling techniques for the system model specification with formal Petri nets, which are used to capture template matchmaking using reachability analysis. This work demonstrates how existing formalisms can be used for the specification of a generic adaptation model for pervasive applications.

Categories and Subject Descriptors: D.2.2 [Design Tools and Techniques]: Petri nets

General Terms: Algorithms

Additional Key Words and Phrases: Cross-layer adaptation, multilayer applications, taxonomies of application mismatches, adaptation templates, matchmaking, context-aware systems, Petri nets

ACM Reference Format:

Popescu, R., Staikopoulos, A., Brogi, A., Liu, P., and Clarke, S. 2012. A formalized, taxonomy-driven approach to cross-layer application adaptation. *ACM Trans. Autonom. Adapt. Syst.* 7, 1, Article 7 (April 2012), 30 pages. DOI = 10.1145/2168260.2168267 <http://doi.acm.org/10.1145/2168260.2168267>

1. INTRODUCTION

Pervasive computing envisions a seamless and distraction-free environment of distributed and heterogeneous applications and devices that utilize resources in their environment. Devices and applications are context-aware, meaning that they can sense changes to their executing environment and manage information automatically and transparently. Recent technological advances in mobile devices as well as wireless and sensor networks make it possible to construct pragmatic, large-scale applications

Authors' addresses: R. Popescu (corresponding author) and A. Staikopoulos, School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland; email: rpopescu@scss.tcd.ie; A. Brogi, Computer Science Department, University of Pisa, Italy; P. Liu and S. Clarke, School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2012 ACM 1556-4665/2012/04-ART7 \$10.00

DOI 10.1145/2168260.2168267 <http://doi.acm.org/10.1145/2168260.2168267>

for pervasive computing. Large-scale applications have the potential to span different infrastructures (wireless networks, sensors, services), combine different technologies (e.g., communications, middleware) and device types, to offer an integrated pervasive framework with rich capabilities across many conceptual application layers. A geographically large-scale crisis management system is illustrated in this article, and is an example of such an application type.

The scale and complexity of such application types pose new challenges. In order to cope with their conceptual complexity, applications will need to be organized in a multilayer architecture. Each layer is likely to use languages and technologies appropriate to its own concerns. For example, there may be layers that manage device types, middleware, organization, behavior, implementation (services), representation, or security-trust concepts of the system. The provision of properties such as robustness and fault tolerance is challenging because of user/device mobility, network vulnerability, and device/software heterogeneity. Typically, such context-aware applications are required to dynamically adapt their structure and behavior based on continuously changing environments and requirements, with such adaptations potentially affecting all defined layers of the application. In addition, the level of heterogeneity inherent in large-scale pervasive applications makes it difficult to foresee all possible types of clients and interaction patterns that such applications must follow. A static adaptation solution encoded in the application is therefore not always sufficient.

In this article we outline a generic formal methodology that supports flexible cross-layer adaptation in multilayer applications. The task of addressing the challenges involved in dynamically adapting multilayer applications can significantly benefit from the use of formal approaches, and this article illustrates the use of different formalisms for different elements of its specification. In particular, we combine semiformal meta-modeling techniques for the structural specification of the model, with the more formal Petri Nets (PNs), which are used to capture complex behavior. The goal is to enhance and semiautomate the adaptation process. At a high level, the main structural elements of the adaptation methodology are: *Templates* (also known as patterns) that define generic adaptation solutions to common application mismatches, and *Taxonomies of application mismatches* that provide classifications of common layer-specific application mismatches [Becker et al. 2004].

We assume that applications monitor, collect, and analyse data received from sensors and the executing environment, validate contextual rules and failures, and trigger the adaptation process by raising events that encapsulate application mismatches [Erradi et al. 2006; Kazhamiakin et al. 2009; Popescu et al. 2009]. At a high level, the main behavior of the adaptation process is the search for adaptation templates that can solve application issues based on existing taxonomies of mismatches. The search involves checking first whether there are templates bound to the mismatch that exactly matches the event. If so, the search continues by investigating these templates' dependents (using template links). Otherwise, the process tries to find adaptation solutions by searching for templates that are bound to ancestor nodes in the taxonomy, up to to root of the taxonomy (we refer to these as "more general" templates). If none is found, then the adaptation process searches for adaptation solutions corresponding to templates that are bound to descendant nodes in the taxonomy (we refer to these as "more specific" templates). Cross-layer adaptation is achieved by linking templates either at the same or at different application layers. Templates may trigger the execution of specific templates through direct invocations, or may raise adaptation events that trigger the matching and execution of other adaptation templates. The matching process inspects such direct and indirect template dependencies to derive sequences of adaptation templates that achieve the cross-layer adaptation needed to solve the mismatch.

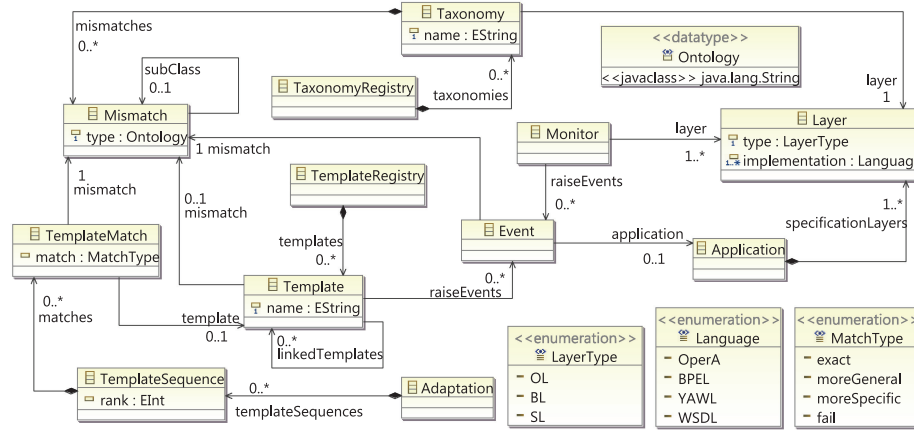


Fig. 1. A metamodel formalization for the system model.

The remainder of this article is organized as follows. Section 2 provides an overview of our cross-layer adaptation methodology. In Section 3 we formalize our methodology and present its application to the adaptation of a pervasive crisis management application. In Section 4 we describe related work. Section 5 presents a qualitative-based evaluation of our approach, followed by some concluding remarks in Section 6.

2. OVERVIEW OF THE ADAPTATION METHODOLOGY

In this section we present the structural and behavioral elements of our adaptation methodology, together with a description of the adaptation process. We illustrate the system model for our approach with a standard MOF [OMG 2006] metamodel; see Figure 1. The metamodel illustrates how elements of our approach such as layers, applications, events, mismatches, taxonomies, templates, template matches, and sequences are defined and used.

Multilayer applications. Our system model supports applications with multiple layers. Each layer has a type and one or more definition or implementation languages. Applications are defined as sets of specification layers. For example, a service-based application may have three layers: a “service layer” specifying services used by the application and defined using WSDL [Christensen et al. 2001], a “behavioral layer” that specifies the application as an orchestration of services and defined using BPEL [Alves et al. 2007], and an “organizational layer” that specifies the stakeholders involved in the business process defined using OperA [Dignum 2003].

Events. Adaptation events encapsulate application mismatches and are raised by layer-specific monitors [Erradi et al. 2006; Ezenwoye and Sadjadi 2007; Kazhamiakin et al. 2009; Moser et al. 2008; Popescu et al. 2009]. Example events may be *message-ordering mismatch* (at a behavioral layer), or *invocation mismatch* (at a service layer).

Taxonomies of application mismatches. We classify adaptation techniques based on taxonomies of application mismatches that they can handle. For each application layer, the application architect/designer defines one or more such taxonomies. Taxonomies should be tree-based with *is a* relationships between children and parent mismatches. Given any two mismatches m_1 and m_2 belonging to the same taxonomy, if m_1 is the same as m_2 we say that they match *exactly*. Both refer to the same application mismatch. If m_1 is an ancestor of m_2 we say that there is a *plug-in* match between them; m_2 is a submismatch of m_1 . Dually, if m_1 is a descendant of m_2 we say that there is a *subsumes* match between them; m_1 is a submismatch of m_2 . Otherwise, there is a *failed* match.

By modeling taxonomies in this way, it is likely that adaptation techniques that can tackle higher-level nodes in the taxonomy can cope with application mismatches at lower levels of the taxonomy. Higher taxonomy nodes refer to bigger adaptation issues that require more radical changes. For example, a mismatch between the expected and actual input of a service could be resolved either by an adaptation that was designed for service input mismatches (exact match) or by an adaptation that was designed for service interface mismatches (more general match). Dually, adaptation techniques that tackle lower-level nodes in the taxonomy may also (partially) solve mismatches at higher-level nodes of the taxonomy. So, for example, a mismatch between the expected and the actual behavior of a service client could be solved by an adaptation that solves sequential mismatches (more specific match), when the service client's behavior is a sequence of service calls.

Adaptation templates. Templates define mechanisms to deal with application mismatches, that is, they express the behavior of adaptation processes. Developers expose adaptation templates as services that provide interfaces for invocation (e.g., WSDL). Developers further associate the templates they develop to application mismatches corresponding to the types of issues they can cope with. For example, an adaptation template based on the algorithm defined in Brogi and Popescu [2006] can be used to solve *message-ordering mismatches* and should be associated to the respective application mismatch. We assume the existence of registries of adaptation templates.

There may be different templates bound to the the same mismatch. For example, one might handle an adaptation in a behavioral layer using BPEL processes, while another adapts YAWL workflows. Depending on the concrete language in use in a mismatch situation, the appropriate template should be used.

Cross-layer adaptation. Application mismatches may require changes at various layers of an application. For example, in our previous service-based application, an event that captures a mismatch between stakeholder roles at the organizational application layer may require the removal of one role and the addition of another. This may also trigger changes at the behavioral and service layers. The behavior may be adapted so as to take into account the new role and a new partner link. A new service may be needed to fulfil an organizational goal of the new role. Such complex scenarios that cross several application layers can be implemented by linking adaptation templates corresponding to layers where adaptation is needed. Templates may be linked both *directly* and *indirectly*. In the direct case, a template invokes another adaptation template. In the indirect case, a template raises an event that will lead to the selection and execution of another template. Linking adaptation templates directly through invocations may be preferred when layers have tight dependencies (e.g., when a behavioral template makes use of another that (un)deploys a service) and these templates are unlikely to change over time, or when linking adaptation templates at the same layer. Linking adaptation templates indirectly through events may be preferred when more flexibility is required, for example, when adaptation templates and application-mismatch taxonomies are likely to change over time, or when triggering templates at different application layers. Linking templates and mismatches in taxonomies has the following benefits.

- Flexibility.* Both application and adaptation logic may evolve over time. As developers evolve their applications, they may replace adaptation logic by replacing the event that triggers the required adaptation, or handle new mismatches by assigning new templates.
- Efficiency.* A hierarchy of application mismatches allows for more tailored adaptations to be performed (whenever available). For example, an event capturing a mismatch in a transport protocol for a service-based application may be tackled more

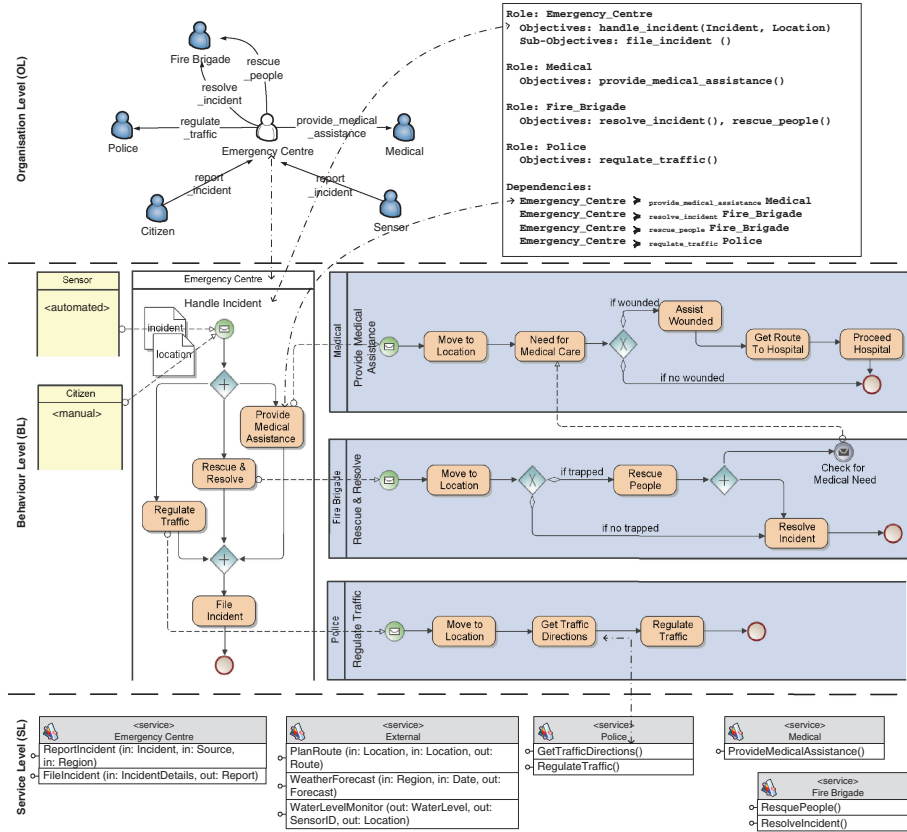


Fig. 2. Layers of a crisis management application.

efficiently through an adaptation template that just replaces the transport protocol, rather than looking for an alternative service and then replacing the entire service. —*Increased application robustness.* Although one would ideally employ only tailored adaptation templates, robust applications may consider employing substitute adaptation templates when no exact ones can be found.

3. FORMALIZING THE ADAPTATION ENVIRONMENT WITH PETRI NETS

The adaptation model can be formally represented by means of PNs [Petri 1962]. More precisely, an *adaptation environment*, consisting of mismatch taxonomies and adaptation templates associated with mismatches, can be modeled by a single PN. Such a PN can be exploited to model which templates can be (best) applied to solve an application mismatch.

Introduction to Crisis Management Case Study. We illustrate the adaptation model with a crisis management application [ALIVE 2010] that consists of multiple layers and must cope with new requirements and failures. The case study explores a flooding incident in the Netherlands, in which a natural disaster has city- or even nation-wide consequences (see Figure 2). Initially, the incident has limited consequences. The stakeholders handling the incident are: the Emergency Center, Police, Fire Brigade, and Medical Agencies. However, as the incident progresses, it is reevaluated and escalated to a more severe level. New requirements emerge and new stakeholders are introduced.

For example, one region has to be evacuated using a Transport Agency. In addition, resources (such as TransportService) may not be interoperable due to a number of mismatches.

The application has three conceptual layers: Organization (OL), Behavior (BL), and Service (SL). The layers are related to one another, so adaptations defined at one layer could be linked to another in any direction. The OL specifies the application's organizational requirements, modeled using OperA [Dignum 2003]. Software entities undertake *roles*, which are assigned to *objectives*, that is, goals that roles have to fulfil. *Dependencies* mark the interactions among roles and depict how *objectives* are fulfilled by using roles. There are six roles: Emergency Center, Police, Fire Brigade, Medical Agency, Citizen, and Sensor (see top part in Figure 2). The Emergency Center role depends on the Citizen and Sensor roles to get an incident report, which initiates its main objective, that is to handle an incident. Handling the incident from the Emergency Center involves regulating the traffic, resolving the incident, rescuing people, and providing medical assistance. These requirements (objectives) are fulfilled by the Police, Fire Brigade, and Medical Agency roles. The BL details how stakeholders (process participants) undertake the organizational roles and orchestrate their tasks to fulfil the organizational requirements: objectives. The orchestration of tasks is represented by the Business Process Modeling Notation (BPMN) [OMG 2009] and executed with a Business Process Execution Language for Web services (WS-BPEL) [Alves et al. 2007]. In our case study, a Citizen (manual task) or a Sensor (automated task) initiates the Handle Incident process provided by the Emergency Center (see middle part in Figure 2). Once the process receives the incident message, a parallel flow initiates the Provide Medical Assistance, Rescue and Resolve and Regulate Traffic tasks, that refer to processes offered by the Medical Agency, Fire Brigade, and Police participants. Once all previous processes terminate, a File Incident task will create an incident report. The SL presents the available services, together with their providers. For example (see bottom part in Figure 2), there are ReportIncident and FileIncident services from EmergencyCenter, GetTrafficDirections and RegulateTraffic services from Police, a ProvideMedicalAssistance service from Medical Agency, RescuePeople and ResolveIncident services from FireBrigade, as well as PlanRoute, WeatherForecast, and WaterLevelMonitor services from other external providers. In principle, tasks defined at the BL are resolved to service invocations.

3.1. Adaptation Templates

Adaptation templates are formally described by PNs. PNs provide a mathematical modeling language for describing distributed systems and process analysis. Formally, PNs are graphs in which nodes are places (depicted as circles) and transitions (depicted as rectangles). Directed graph arcs connect places and transitions. Arcs originate at places and target transitions, or vice versa. Places hold tokens (e.g., a token can represent a condition such as “the adaptation environment has a template associated with mismatch m ”). Transitions “transport” tokens from their input to their output places. Transitions “fire” when all their input places contain tokens and consume a token for each input arc. When transitions fire, they produce tokens for each of their output links. PNs representing the adaptation environment support finding paths from an input mismatch node (matching a triggered adaptation event) to target adaptation templates. The adaptation environment PN first directs the search for adaptation templates bound to the mismatch that matches a triggered event (“exact”). If no such templates exist, the PN directs the search for templates bound to ancestors of the matched mismatch (“more general”) and, if no such templates exist, the PN directs the search for templates bound to descendants of the matched mismatch (“more specific”). Further information on PNs can be found in Murata [1989].

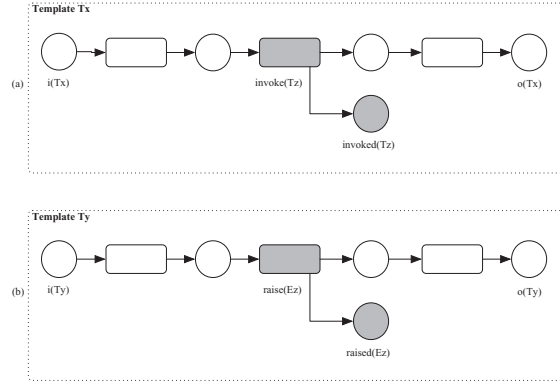


Fig. 3. Example templates: (a) Template Tx invokes another template Tz . (b) Template Ty raises an event Ez . (Key transitions and places are colored in grey.)

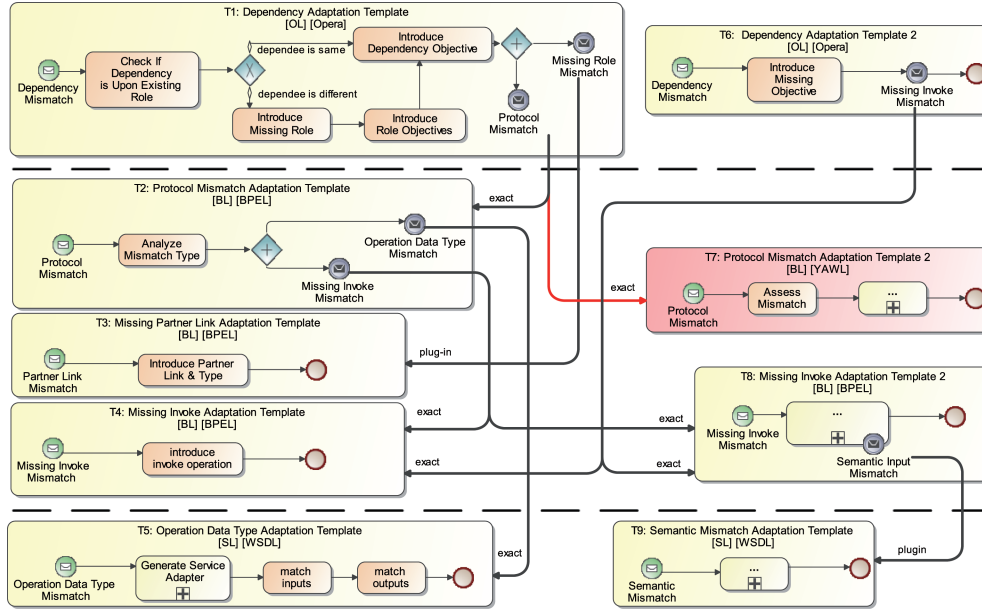


Fig. 4. Examples of adaptation templates with potential matches.

Templates that invoke other templates and/or raise events are modeled by PNs that describe their nondeterministic (communication) behavior. Figure 3 shows the PN specifications of two examples. The top shows a template Tx that invokes another template Tz (direct link). The bottom shows a template Ty that raises an event Ez (indirect link). An invoked template is defined with an *invoke* transition followed by an *invoked* place. A raised event is defined with a *raise* transition followed by a *raised* place.

Adaptation Templates for the Crisis Management Case Study. The adaptation environment for this application includes the templates represented in Figure 4. The templates are represented in BPMN [OMG 2009] form. In this figure, we draw attention to one example of an exact match and one example of a more general match. At

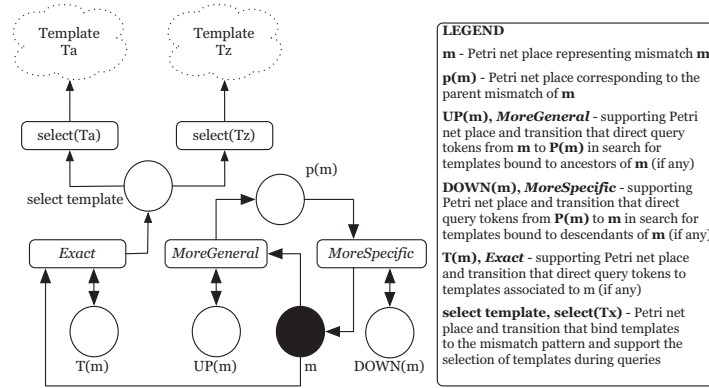


Fig. 5. PN taxonomy pattern.

the OL (top of figure), the adaptation template T_1 handles a *dependency mismatch* for the OperA language. T_1 will initiate two parallel mismatches: a *protocol mismatch* and a *missing role mismatch*. The protocol mismatch is matched exactly to triggers of T_2 and T_7 . However, T_7 is not a valid match as it is associated with a different implementation language (YAWL), so it is ignored. The missing role mismatch is matched to a partner-link mismatch of T_3 with a more general match. Across the three layers, mismatch triggering occurs in a similar manner. Please refer to Section 2 of the electronic appendix for details on the mismatch taxonomies of this scenario.

3.2. Taxonomies of Application Mismatches

We employ a PN pattern for the formal description of mismatches (see Figure 5). The pattern defines PN places and transitions that encode the following behavior.

- Select an adaptation template.* In an adaptation environment, adaptation events are represented as tokens placed in the application mismatches they match. For example, an event of type m (that occurs at application runtime) will be represented by a token in the m place of the corresponding mismatch. The availability of an adaptation template bound to this mismatch (i.e., an adaptation solution for mismatch m) is represented by a token in the $T(m)$ place. The PN pattern will enable the execution of a *select(Tx)* transition that will lead to the selection of an adaptation solution for m .
- Navigate the taxonomy of mismatches upwards.* The *MoreGeneral* transition and $UP(m)$ place enable the search for more general adaptation solutions when no adaptation templates are bound to m . In such cases a token will be pushed from m into the place representing its parent mismatch in the taxonomy, $p(m)$.
- Navigate the taxonomy of mismatches downwards.* Assuming an adaptation event matching $p(m)$, the *MoreSpecific* transition and $Down(m)$ place enable the search for more specific adaptation templates when no adaptation templates are bound to $p(m)$ or to any of $p(m)$'s ancestors.

The algorithms defining the required PN changes for the association and dissociation of templates are presented in Section 3 of the electronic appendix.

Adaptation Environment for the Crisis Management Case Study. In this scenario, the adaptation environment is composed of three taxonomies, one for each application layer. Due to space restrictions, in Figure 7 we depict just a part of the adaptation environment for the crisis management case study.

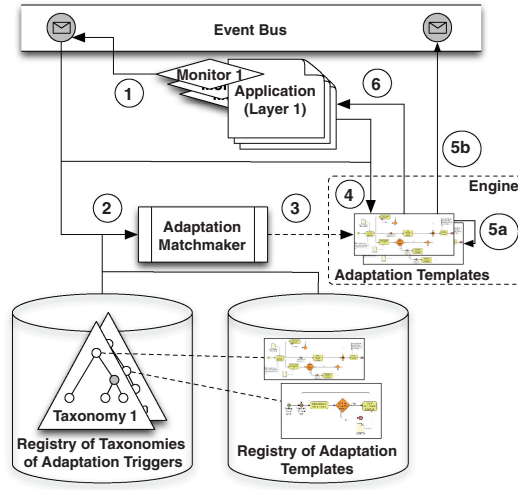


Fig. 6. Overview of our adaptation methodology.

3.3. Adaptation Process

Figure 6 illustrates the main steps of our methodology for the cross-layer adaptation of multilayer applications.

3.3.1. Triggering the Adaptation Process. The adaptation process starts when a layer monitor (or a human stakeholder) raises an adaptation event (step 1 in Figure 6). A matchmaker selects adaptation templates that may tackle the application mismatch identified by the event (steps 2 and 3 in Figure 6). The matchmaker inputs a query that references the event, application, and adaptation environment and outputs a ranked list of *sequences of templates* that may solve the mismatch. The process first identifies the taxonomy and mismatch that match the event. Then, it “navigates” the taxonomy in search of adaptation templates associated to the respective mismatch (*exact* solutions), their ancestors (*more general* solutions) or descendants (*more specific* solutions). Sequences of templates result from this process. The core of the matchmaking process is a reachability analysis of the PN that represents the adaptation environment.

Selecting the taxonomy and mismatch. The event definition contains a reference to a taxonomy mismatch. This information is used by the matchmaker to identify the taxonomy and the mismatch in the adaptation environment of the application. If the mismatch is found, then the matchmaker enables the search for possible sequences of adaptation templates by placing a token in the PN place corresponding to the matched mismatch. If no mismatches are found, the adaptation process aborts.

Matching adaptation templates. The matchmaker checks whether there are templates bound to the matched mismatch. For each such template, the matchmaker checks whether the template can be employed for the desired adaptation by verifying that the specification of the application at the matched taxonomy layer can be processed by the template. For example, for our crisis management case study, templates bound to mismatches at the behavioral layer have to be able to adapt BPEL processes [Alves et al. 2007], since the application’s behavioral layer is expressed using BPEL. Next, the matching process analyses the template’s dependencies as directed by the PN template structure. If no *exact* templates are found, the matchmaker checks whether there are any templates corresponding to the parent of the matched mismatch. If any exist, the adaptation process continues by analysing their dependencies (viz., links to

other templates or taxonomies) as described previously. Otherwise, the matchmaker continues by searching for templates corresponding to the rest of the ancestors of the matched mismatch (if any), up to the root mismatch.

If there are no templates bound to the root mismatch, the matchmaker investigates the children of the matched mismatch. If any exist, the adaptation process analyses their dependencies. Otherwise, the matchmaker searches for templates corresponding to the rest of the descendants of the matched mismatch (if any), down to the leaves of the taxonomy.

The matchmaker generates the possible sequences of templates that may tackle a raised event through a reachability analysis of the PN encoding the adaptation environment. Please refer to Section 4 of the electronic appendix for a description of how the reachability analysis works.

Ranking sequences of templates. The matchmaker allows for a user-configurable ranking of matched sequences of adaptation templates that employs the following criteria:

- C_1 . number of more specific template matches,
- C_2 . number of more general template matches,
- C_3 . number of templates, and
- C_4 . number of raised events.

Generally speaking, *exact* templates are preferred to *more general* templates, which are preferred to *more specific* ones. Note that, from a ranking perspective, templates invoked directly are treated as *exact* templates. By default, the ranking system applies the aforesaid criteria in top-down order and sequences with lower criteria scores rank higher. When two sequences have the same score for criterion C_x , criterion C_{x+1} applies (if any). Hence, a set of three sequences $SequenceSet = \{S_1, S_2, S_3, S_4\}$, where $CriteriaScores(S_1) = \langle 2, 0, 3, 1 \rangle$, $CriteriaScores(S_2) = \langle 0, 0, 3, 3 \rangle$, $CriteriaScores(S_3) = \langle 0, 1, 4, 3 \rangle$, and $CriteriaScores(S_4) = \langle 0, 1, 5, 2 \rangle$ will be ranked as “ S_2, S_3, S_4, S_1 ”, with S_2 ranked highest. The criteria to be applied, and their order, can be configured by the user.

Adaptation Event and Template Sequences for the Crisis Management Case Study. As the case study is executed, contextual rules are evaluated periodically by the monitor by collecting data from sensors and services. As the severity of the incident escalates, these rules are validated to true and specific events are emitted and matched against predefined adaptation templates to accommodate the new requirements and failures. For example,

if(WaterLevel > 10m and Forecast = Storm) then adapt(Level3)

states that the case study will be adapted to a new severity level (Level3), if the WaterLevelMonitor service and the WeatherForecast service report that the WaterLevel is greater than 10m and the Forecast is storm, respectively. Consequently, processes that are still active and long running such as the Handle Incident (Provide Medical Assistance, Rescue and Resolve, and Regulate Traffic) need to be adapted to reflect the new configuration requirements. The increase in the incident’s severity mandates that the Emergency Center has to coordinate the evacuation of the region, by utilizing a new role, Transport Agency. A *dependency mismatch* event is emitted from the monitor, denoting the evacuate dependency *objective* and the Transport Agency role are not fulfilled by the Emergency Center role (Figure 8). The matchmaker will fetch and match the event to a *dependency mismatch*. This is an exact match that can trigger both T_1 and T_6 templates, defined at the OL (Figure 4).

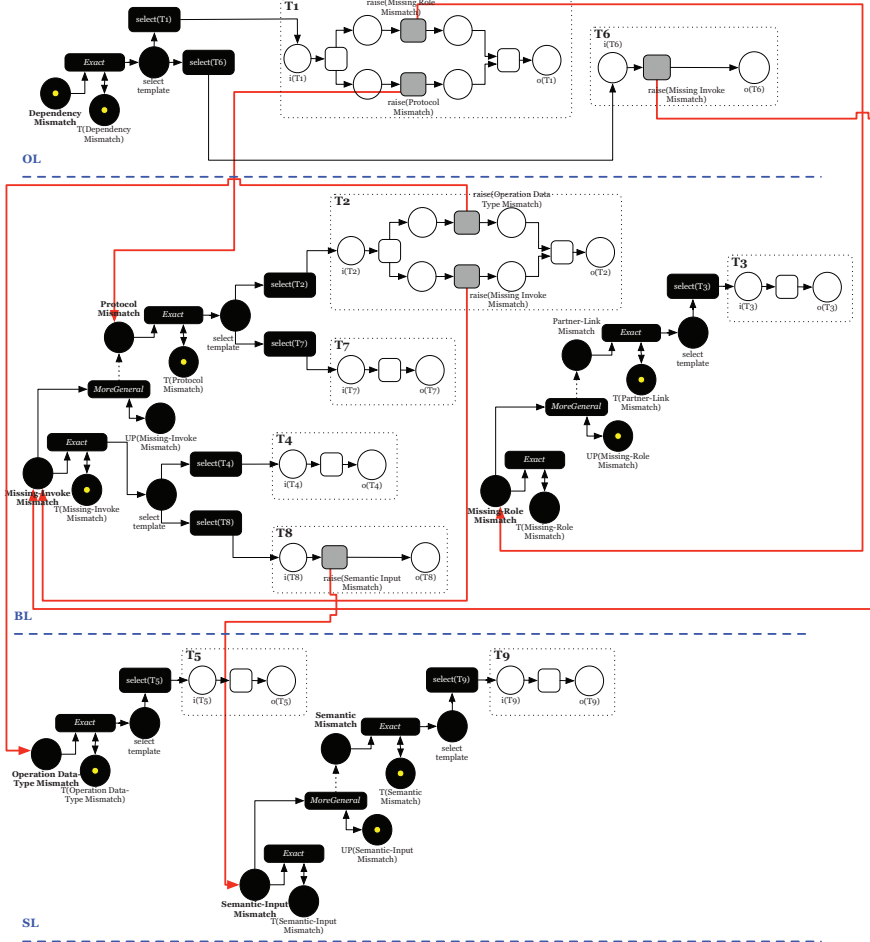


Fig. 7. Adaptation environment for the crisis management case study. (Taxonomy transitions and places are colored in black. Transitions that invoke templates or trigger events are colored in grey. Tokens are colored in yellow.)

From the configuration of adaptation templates, different adaptation template sequences can be resolved. Selecting the more suitable one is based on an analysis and ranking of the criteria, in this case starting from the adaptation environment in Figure 7. With an event of type *dependency mismatch*, we can have the following valid sequences, ranked as follows.

- (1). $\langle T_6, T_4 \rangle$, CriteriaScore: $\langle 0, 0, 2, 1 \rangle$ and
- (2). $\langle T_6, T_8, T_9 \rangle$, CriteriaScore: $\langle 0, 1, 3, 2 \rangle$.
- (3). $\langle T_1, T_3, T_2, T_5, T_4 \rangle$, CriteriaScore: $\langle 0, 1, 5, 4 \rangle$, and
- (4). $\langle T_1, T_3, T_2, T_5, T_8, T_9 \rangle$, CriteriaScore: $\langle 0, 2, 6, 5 \rangle$.

PN transitions executed on a reachability graph corresponding to the adaptation environment in Figure 7 are: *Exact*(*DependencyMismatch*), *select*(*T6*), *raise*(*MissingInvokeMismatch*), *Exact*(*Missing – RoleMismatch*), and *select*(*T4*), from which sequence (1), $\langle T_6, T_4 \rangle$, was extracted.

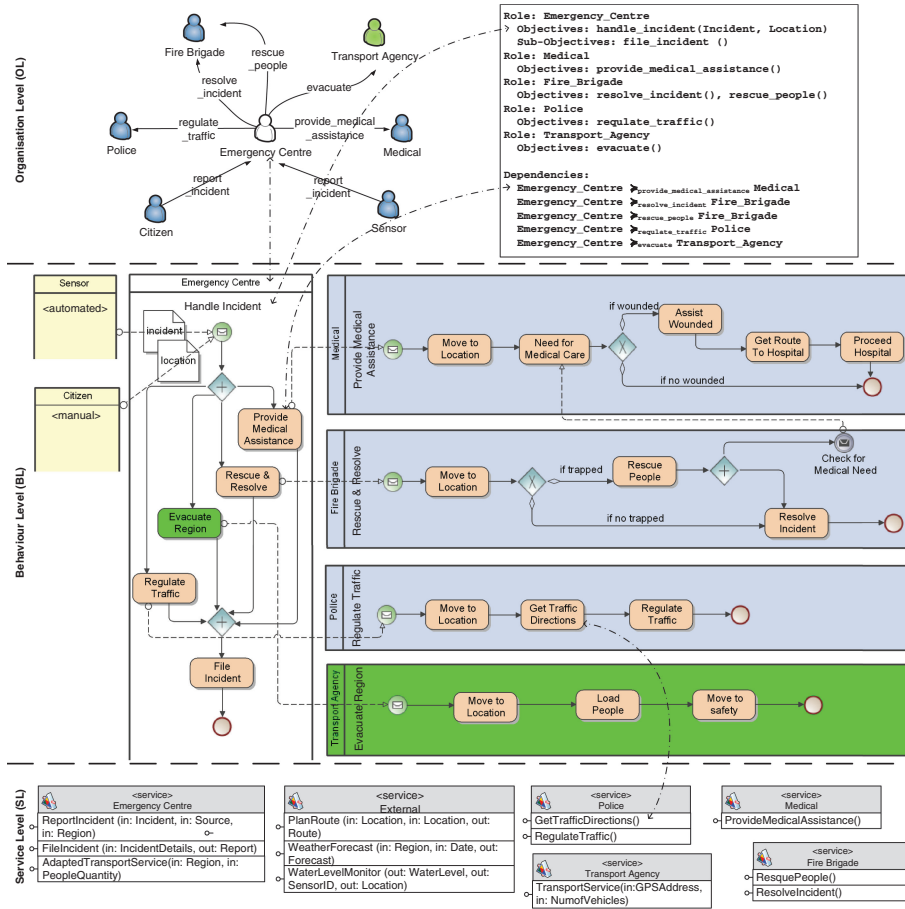


Fig. 8. Adapting the case study across layers due to reconfiguration.

3.3.2. Performing the Adaptation. The process of engineering and adapting applications may be fully automated when the sequences of adaptation templates contain only *exact* templates for which there exists the necessary input information (e.g., the inputs needed for the execution of the templates are provided by the raised event). Sequences containing *more general* and *more specific* templates require developer intervention for adaptation selection. The developer may choose an adaptation based on the sequences presented and may also customize it prior to its execution based on whether it can handle the required adaptation given the application execution context.

Adapting the Crisis Management Case Study. Figure 8 presents the result of an adaptation process based on sequence (3), which provides cross-layer adaptation across all application layers. At the OL, a new evacuate *objective* is assigned to the Emergency Center, using a new Transport Agency role. At the BL, an evacuate task is introduced in the Handle Incident process using a service from the Transport Agency. At the SL, the Transport service is adapted by the AdaptedTransport service that resolves parameter types mismatches.

More specifically, the application of the T_1 adaptation template introduces a new role (Transport Agency) to offer transport for evacuation. For the missing role (Transport Agency), the application of T_3 produces missing partner links and types, so

TransportService can be invoked. The assignment of the evacuate *objective* dependency also triggers a *Protocol mismatch*. T_2 identifies both a missing invoke and an *Operation Data Type mismatch*. The missing invoke initiates the T_4 adaptation template that alters (Handle Incident) by inserting an invoke operation for the evacuation service provided by the Transport Agency. The *Operation Data Type mismatch* is triggered by incompatible service parameters. The TransportService requires as input a GPSAddress and a number of vehicles, where previously the application operates with Regions and people numbers. The application of the T_5 adaptation template creates a proxy service adapter (AdaptedTransportService), that intercepts the invoke operation and converts the service parameters to appropriate types.

4. RELATED WORK

This section presents related core adaptation techniques, aspect-oriented programming-based adaptation approaches, ontology-aware adaptation frameworks, formal (PN-based) adaptation frameworks, and multilayer adaptation approaches.

Core adaptation frameworks. Targeted only at the service layer, Erradi et al. [2006] describe a framework for dynamic Web service selection and composition, designed to improve service application dependability. Dedicated framework services monitor interactions with participating services to verify that monitoring policies are satisfied. Similarly to our approach, whenever an undesired condition is detected, the monitoring service generates a violation event to trigger adaptation. van den Heuvel et al. [2007] propose a configurable adapter architecture for self-adaptive Web services. The key construct is a generic protocol adapter that defines a mapping between businesses' protocols that orchestrate service providers and consumers. At runtime, a service manager composes existing mappings to adapt interacting services. Although the architecture is extensible, it solves only Web Service protocol mismatches and it is not clear how self-adaptation of Web services can be triggered and woven to the running business process instance.

Aspect-oriented programming-based approaches. Various approaches propose the use of Aspect-Oriented Programming (AOP) to implement an adaptation. Kongdenfha et al. [2009] propose a framework using AOP for service adaptation due to interface and protocol mismatches. The approach requires developers to manually define a mismatch before performing the adaptation. Charfi et al. [2009] define a plug-in architecture for self-adaptive Web service compositions by modularizing self-adaptation features in aspect-based plug-ins. Aspects can be hot-deployed to BPEL engines that support the aspect-oriented workflow language AO4BPEL. Karastoyanova and Leymann [2009] illustrate how the AOP paradigm can be mapped and applied in the BPEL language to enable the adaptation of running orchestrations. The authors do not discuss how to generate appropriate WS-Policy attachments (aspects) for the desired service adaptation.

Ontology-aware adaptation frameworks. There is also ongoing research on automatic matchmaking and adaptation of Web services using ontologies. William et al. [2003] provide a framework for semantic matchmaking and service adaptation called ICENI. In this framework, the programmatic interface of services is annotated using OWL [McGuinness and van Harmelen 2004] in relation to some domain concepts. With the help of this ontology information, a syntactically different but semantically equivalent service can be autonomously adapted and substituted, but it deals only with signature mismatches. Syu [2004] proposes ontology-aware approach service adaptation that solves limited cases of service signature mismatch and returns only exact matches.

Formal adaptation frameworks. Zhang and Cheng [2006] propose a model-driven process for the development of dynamic adaptive software. PNs are used to model adaptive components generated from high-level requirements, and can be used to generate executable adaptive programs. Canal et al. [2008] describe the automatic generation

of adaptation contracts used to overcome signature and behavioral mismatches. The algorithm is based on synchronous products and PN encoding. Gierds et al. [2008] generate an adapter for interacting services with mismatches. Based on the Specification of Elementary Activities (SEA) that consists of transformation rules on message types that services use, the adapter specification can be generated automatically using a PN algorithm. Martens [2005] presents a method to define and verify usability (i.e., a soundness criterion for business workflow modules) using PNs.

With respect to these adaptation approaches, our proposal features a formal cross-layer adaptation framework that handles generic multilayer applications using taxonomies of mismatches and adaptation templates. Our focus is on the dynamic and flexible discovery of composite adaptation templates that solve cross-layer adaptation dependencies.

Multilayer adaptation approaches. Gjørven et al. [2008] propose a technology-agnostic adaptation middleware that can be used to integrate adaptation techniques from both application and service layers. The middleware focuses on providing a framework that integrates adaptation techniques in different layers in order to control them in one place. Kazhamiakin et al. [2009] propose a cross-layer framework and underlying conceptual model adopted in the S-Cube [2011] project to address monitoring and adaptation in service-based applications. The authors provide a set of requirements for cross-layer monitoring and adaptation frameworks of service-based applications and illustrate a uniform conceptual model underlying such frameworks. However, this cross-layer framework, to the best of our knowledge, is bound to three predefined layers (i.e., BPM, SCC, and SI), and new layers cannot be easily supported.

In summary, many existing adaptation techniques, such as the ones presented before, can be plugged into our adaptation framework as adaptation templates. For example, one may define an adaptation template based on the algorithm described in Kongdenfha et al. [2009] to solve *extra-message mismatches* at the behavioral layer.

Ontology-aware service discovery approaches. Related work in semantic Web service discovery matches services to client queries. Some approaches search for single services (e.g., Li and Horrocks [2003], Klusch et al. [2006], and Srinivasan et al. [2006]) while others combine multiple (partial) service matches into composite services (e.g., Aversano et al. [2004], Benatallah et al. [2005], and Mokhtar et al. [2005]). These approaches, however, do not offer direct support for software adaptation. Matches usually consist of services that comply with the inputs and outputs specified by the query. Bansal and Vidal [2003] and Pagliarecci et al. [2007], among others, augment queries with a specification of the desired service behavior. Query inputs and outputs can contain ontological information, which can be used (similarly to Paolucci et al. [2002]) to match services that can provide alternative outputs, or accept alternative inputs. For more information on ontology-aware service discovery approaches, see Corfini [2008].

5. EVALUATION

In this section, we study the impact on the overall development lifecycle for our approach as compared to related adaptation frameworks and approaches. Through this study, we assess the development effort (main development tasks) needed, the Required Knowledge and technology (R.K.), the Development Expertise (D.E.) and the Time Effort (T.E.). For this assessment, we employ a number of case studies on a crisis management application and an automotive application, to illustrate how each copes with adaptation scenarios to: modify existing adaptations to new requirements (Section 5.1); add new adaptations (Section 5.2); and add new layers to existing applications (Section 5.3).

We compare our approach (which we call CLAMS) to three of the closest state-of-the-art approaches from the Related Work section. For the evaluation, we use applications from two different domains: the crisis management system described previously, and a case study from the automotive domain taken from S-Cube, the European Network

of Excellence in Software Services and Systems [S-Cube 2011]). Our assessment of each approach's features is synthesised (in Section 5.4) in a set of results that reflect standard software development process steps, used as the basis for comparison. These are: requirements analysis (Req), Design (Des), Implementation (Imp), Integration (Int), Testing (Tes), Deployment (Dep), and Maintenance (Mai). For all scenarios and frameworks considered, we assume the existence of adaptation behavior (available as WSDL services) from which templates can be (manually) generated. The availability, or lack thereof, of code to support such behavior will have the same development impact for every approach considered.

5.1. Modify an Existing Adaptation to New Requirements

In this case we evaluate how an existing adaptation environment can be reused or modified to an emerging adaptation requirement. In particular, we distinguish two cases: one in which the existing adaptation environment is sufficient to adapt the application to its new requirements and one in which the adaptation environment needs to be modified.

In the crisis management scenario different stakeholders need to be orchestrated for the evacuation of a region when the threat level increases above a certain threshold. The application consists of three conceptual layers: Organization (defining relationships and rules among stakeholders in OperA), Behavior (defining stakeholders' processes in BPEL), and Service (defining services offered or used by stakeholders in WSDL). Please note that we abbreviate the three application layers of this scenario as OL, BL, and SL.

First, we assume that the medical team also needs to provide assistance to remote locations. In this case, the wounded are transferred either directly to hospitals as before, or to a nearby location for transfer with an air ambulance. The required adaptation is performed at the BL, by modifying the *Provide Medical Assistance* process for a medical team. As a result, a conditional node is introduced to check whether air assistance is needed, by raising a *Sequence vs. Conditional Mismatch*, followed by either the existing task to *Get Route to Hospital* or a new task to *Proceed to Nearby Air Ambulance* by raising a *Missing Invoke Mismatch*. In this case, the adaptation environment remains unchanged.

Second, we assume that developers want to enhance the adaptation environment to solve semantic mismatches of inputs and outputs of services, for example, to identify locations as regions with GPS coordinates. This is achieved by modifying the *Operation Data Type* adaptation template by either associating it directly with the *Semantic Mismatch* adaptation template, or by triggering a *Semantic Mismatch*. In this case, the adaptation environment is updated automatically to reflect changes of the adaptation templates.

Table I shows that most approaches require a lot of effort to maintain and validate the adaptation environment, as they are defined manually and they are not supported by automated tools or model abstractions. Our approach, CLAMS, requires less effort as it has been specifically designed to support modifications of the adaptation environment in a flexible and dynamic way. In addition, the adaptation environment is generated automatically and verified by validating its formal PN representation.

5.2. Adding a New Adaptation at Behavioral Layer

In this scenario, we assume that a new adaptation template T_{10} has been deployed in the case study described in Section 3, which solves the *Missing-Role Mismatch* with an *exact* match. Here, CLAMS replaces the more general template T_3 with the exact template T_{10} in the previous adaptation sequence $\langle T_1, T_3, T_2, T_5, T_4 \rangle$.

Table II summarizes the tasks needed to achieve the inclusion of the new adaptation into the existing adaptation environment and the generation of a new cross-layer

Table I. Modify an Existing Adaptation to New Requirements

	Developer Tasks	R.K.	D.E.	T.E.
CLAMS	T1) If needed modify existing adaptation templates by invoking either new services or other available templates, or by raising <i>Mismatches</i> .	SoC, template design	M	M
	T2) Verify that the adaptation environment is correctly modified. Correctness is based on an automated PN analysis.	PN, automated, tool based	L	L
	T3) Implement Mismatch trigger. Implement Web service client code to trigger the appropriate <i>Mismatch</i> event.	SoC, HPL	M	M
	T4) Test adaptation. Automatically simulate <i>Mismatches</i> to match adaptation sequences; and check desired adaptation happens.	Domain Knowledge, automated	L	L
S-CUBE	T1) Identify adaptation strategies to solve Mismatches. Identify appropriate adaptation strategies for <i>Mismatches</i> .	Domain Knowledge	H	H
	T2) Update/Implement adaptation. Update framework with new triggering events, maintaining relationships with other events and adaptation strategies.	SoC, HPL	L	M
	T3) Implement the Mismatch trigger. Convert original monitoring event emitted to required format.	SoC, HPL	L	M
	T4) Test adaptation. Validate cross-layer adaptation activities.	XML	L	L
Kongdenfha	T1) Identify and integrate behavioural-layer adaptations. Identify BL mismatch patterns and instantiate adaptation templates.	SoC, AOP, BPEL	M	M
	T2) Identify and integrate service-layer adaptations. Identify SL mismatches and instantiate adaptation templates.	SoC, AOP, BPEL	M	M
	T3) Integrate T1) - T2) into cross-layer adaptation. Integrate and coordinate adaptation mechanisms across two application layers.	SoC	L	H
	T4) Test adaptation. Adaptation aspects are deployed to the ActiveBPEL engine. Check expected adaptation target is achieved.	XML, ActiveBPEL	M	L
Erradi	T1) Modify the monitoring and adaptation policies (<i>no support</i>). Use WS-Policy4MASC to manually modify monitoring and adaptation policies.	WS-Policy 4MASC, SoC	H	H
	T2) Identify BL adaptation policies as required in T1).	WS-Policy 4MASC, SoC	H	M
	T3) Integrate and coordinate policies chosen in T1) and T2) to enable cross-layer adaptation (<i>no support</i>). Manual orchestration process.	WS-Policy4 MASC, SoC	H	M
	T4) Test adaptation. Validate cross-layer adaptation policies composed in T3).	MASC middle-ware, SOAP	H	L

Legend:

- H: High requirement; M: Medium requirement; L: Low requirement;
- SoC: Service-oriented Computing technologies; PN: Petri net;
- HPL: High-level programming language; MASC: Manageable and Adaptive Service Compositions.

adaptation. From the comparison it is apparent that all approaches require medium to high level of expertise and effort to implement a new adaptation using a specific model/syntax. In CLAMS, this can be achieved more easily by associating the new adaptation template to a certain mismatch and linking it to other templates through triggered events, or direct invocations. The adaptation environment is also updated automatically, thus the effort needed to integrate the new adaptation is significantly less. In addition, while in other approaches a newly added adaptation can only serve as an exact match, in CLAMS, it can be resolved to an exact, a more general, or a more specific match, providing a high level of flexibility. Finally, the effort needed for triggering and testing the new adaptations is relatively low for all approaches, as it mostly requires developing and testing Web service clients.

5.3. Adding a New Layer to the Automotive Manufacturing Application

In this case, we illustrate the process of updating the adaptation environment of an application, due to the addition of a new layer. In particular we consider the

Table II. Adding a New Adaptation at Behavioral Layer

	Developer Tasks	R.K.	D.E.	T.E.
CLAMS	T1) Convert the new adaptation to a PN representation and integrate with the existing adaptation environment .	Domain Knowledge, PN	M	M
	T2) Test adaptation. Trigger a <i>Dependency Mismatch</i> to match adaptation sequences and check if new template, <i>T10</i> , is selected and execution of selected adaptation sequence performs desired adaptation.	XML	L	L
S-CUBE	T1) Implement a new adaptation strategy. Implement new adaptation strategy to solve a <i>Missing-Role Mismatch</i> .	Domain Knowledge, SoC, S-Cube model	H	H
	T2) Link the new adaptation strategy to events and adaptation strategies in other layers. Explicitly relate new adaptation to corresponding monitoring events and strategies in other layers.	SoC, HPL	L	M
	T3) Test adaptation. Trigger <i>Dependency Mismatch</i> to validate that new adaptation tackles <i>Dependency Mismatches</i> and adaptation is performed in all three layers.	XML	L	L
Kongdenfha	T1) Identify a new <i>Missing-Role Mismatch</i> pattern if there is no existing one. Identify adaptation patterns and provide template to resolve mismatch.	SoC, AOP, BPEL	M	H
	T2) Instantiate the <i>Missing-Role Mismatch</i> template. Instantiate adaptation template included in new <i>Missing-Role Mismatch</i> pattern using new adaptation strategy.	SoC, AOP, BPEL	M	M
	T3) Integrate the new <i>Missing-Role Mismatch</i> template into cross-layer adaptation (no support). Integrate (manually) the new <i>Missing-Role Mismatch</i> template in response to a <i>Dependency Mismatch</i> .	SoC, BPEL	M	H
Erradi	T1) Implement a new Event-Condition-Action adaptation policy to solve the <i>Missing-Role Mismatches</i> .	WS-Policy 4MASC, SoC	H	H
	T2) Integrate the new policy to generate cross-layer adaptation (no support). Manual orchestration activity.	WS-Policy 4MASC, SoC	H	H
	T3) Test adaptation. Validate that cross-layer adaptation policies composed in T2) and deployed, fulfill the new objective.	MASC middle-ware, SOAP	H	L

automotive manufacturing case study published by S-Cube [Kazhamiakin et al. 2009], which explores the simulation and analysis of new automobile models before moving to mass production. Typically, after the design of a new automobile model, engineers would conduct a computer-based simulation on the new model to reproduce the characteristics of real vehicles.

The original automotive manufacturing application consists of three conceptual layers: BL, SL, and SI. A new Security Layer (SecL) should be added, to enhance the privacy policy that simulation data must be kept at servers with access by authorized entities only. In addition, any violation of the security rules specified in the SecL may lead to an adaptation performed in different layers, to ensure compliance with the security policy.

The problem the developers face is that a new layer and corresponding adaptation mechanisms should be integrated into the existing adaptation environment in a systematic way to ensure new cross-layer adaptation is generated correctly and effectively. In this scenario, we assume that mismatch taxonomies and adaptation templates for the SecL are already available in the CLAMS registry. This assumption is reasonable as in the long term, it is more likely for developers to find reusable taxonomies and templates for their applications.

Table III summarizes the tasks needed to achieve the inclusion of the new adaptive behavior for the SecL into the automotive manufacturing application. From the comparison it appears that the effort required by CLAMS to design, implement, and integrate adaptation logic into new application layers is significantly less when we

Table III. Adding a New Layer to the Automotive Manufacturing Application

	Developer Tasks	R.K.	D.E.	T.E.
CLAMS	T1) Update the existing adaptation environment to include taxonomies and templates for the SecL. Search for taxonomy and adaptation templates for the SecL in the CLAMS registry; add the PN representation to existing adaptation environment.	Domain Knowledge, PN	M	L
	T2) Verify adaptation environment to perform required cross-layer adaptation. Analyse the PN to ensure that the adaptation environment can generate adaptation sequences to solve privacy policy violation and that adaptation environment is deadlock-free.	PN	M	M
S-CUBE	T1) Identify adaptation strategies to solve the data privacy violation in the SecL layer. Either identify reusable adaptation strategies in the public knowledge domain or develop one from scratch.	Domain Knowledge, SoC, HPL	H	H
	T2) Implement cross-layer adaptation. Link new adaptation strategies added in the SecL to those in different layers to generate coordinated cross-layer adaptations.	SoC, HPL	L	M
	T3) Test adaptation. Check if cross-layer adaptation activities performed in all layers to solve the data privacy violation.	XML	L	L
Kongdenfha	T1) Identify new adaptation patterns and templates in the SecL (<i>no support</i>). Developers need to identify new patterns and corresponding adaptation template in the SecL layer.	Domain Knowledge, adaptation pattern, AOP	H	M
	T2) Integrate new adaptation templates into cross-layer adaptation (<i>no support</i>). Manual integration activity.	SoC	L	H
	T3) Test adaptation. Deploy adaptation aspects to the ActiveBPEL engine. Check any violation of data privacy rules in the SecL will lead to cross-layer adaptations.	XML, ActiveBPEL	M	L
Erradi	T1) Specify monitoring and adaptation policies that tackle data privacy violation in the new SecL layer (<i>no support</i>). Developers use the WS-Policy4MASC language to manually specify the policies.	WS-Policy 4MASC, SoC	H	H
	T2) Integrate and coordinate policies developed in T1) with existing ones to generate cross-layer adaptation (<i>no support</i>). It requires the manual orchestration of adaptation policies across the application layers.	WS-Policy 4MASC, SoC	H	H
	T3) Test adaptation. Developers validate if any violation of the data privacy rules as specified in the SecL would trigger adaptation policies in different layers to restore the compliance to the security policy.	MASC middle-ware, SOAP	H	L

assume the reuse of layer taxonomies and adaptation templates. Similarly to other scenarios, other approaches require more effort from developers, who have to manually define, implement, and integrate cross-layer adaptation solutions.

5.4. Discussion

Table IV summarizes the level of support offered by CLAMS and related approaches to the development process of cross-layer adaptations. The scores in the table take into account the features of each approach and the developer effort required.

From our analysis, we conclude that CLAMS provides better support for the rapid design, implementation, integration, and maintenance of cross-layer adaptation into multiple-layer applications. The main factors that justify this conclusion are as follows.

- Design*: extensible registries of taxonomies and templates, reusable taxonomies and templates (across same or different application domains), formal verification of the adaptation environment, alternative template matches that provide flexible adaptations through more general and more specific solutions,
- Implementation*: registries of reusable taxonomies and adaptation templates (exposed as WSDL services), which provide implemented adaptation solutions,
- Integration*: the possibility to specify (loosely coupled) cross-layer adaptations through events or direct template invocations,

Table IV. Summary of Support for the Software Development Process

Approach	Req	Des	Imp	Int	Tes	Dep	Mai
CLAMS	0	4	3	3	1	3	4
S-Cube	0	3	2	2	1	2	2
Kongdenfha et al.	0	2	2	2	1	2	2
Erradi et al.	0	1	2	1	1	2	2

Legend: 0 (no support) → 1 (low) → 2 (low-to-medium) → 3 (medium) → 4 (medium-to-high) → 5 (high)

- Deployment*: registries of reusable taxonomies and adaptation templates (exposed as WSDL services), which provide deployed adaptation solutions (“on-demand adaptation services”), and
- Maintenance*: all of the preceding.

Where registries of taxonomies and templates are available, the process of developing cross-layer adaptations will be simplified through taxonomy and template reuse, loosely coupled adaptations, and the scope to provide alternative adaptations.

6. CONCLUSION

Large-scale, pervasive applications are required to dynamically adapt to cater for changes to the environments in which they execute. Given the scope and spread of architectural possibilities for pervasive application specification, it is challenging to provide a flexible adaptation model that can be utilized for an unpredictable range of architectural-layer types, and that caters for adaptations that may, in a single instance, affect multiple layers. In this article, we have outlined a generic methodology for cross-layer adaptation in multilayer applications. The methodology’s main structural ingredients are events, taxonomies of application mismatches, and adaptation templates. As we would expect, the process of capturing these structural elements in a semiformal metamodel enabled us to reason about the relationships between the elements, and clarified our thinking on their specification. In addition, the common representation enabled with metamodels allows us to reason about integration properties of heterogeneous adaptation environments.

From a behavioral perspective, we were obliged to consider a more formal approach. The level of analysis needed to ensure detection of template dependency cycles, or deadlocks, and to support reachability analysis, led us to utilize PNs for the formalization of the matchmaking process. This process employs taxonomies of application mismatches to dynamically select adaptation templates based on the degree of match between their triggering mismatches and a raised event. Flexibility is achieved by matching templates corresponding to plug-in (more general) and subsumes (more specific) mismatch matches. Cross-layer adaptation is achieved by allowing templates to be linked both directly, through invocations, and indirectly, through events. A PN encodes application mismatch taxonomies and template dependencies and this enabled us to solve matchmaking queries by investigating possible template sequences obtained through a reachability analysis. A similar analysis may be employed to detect template dependency cycles, or deadlocks.

Issues that remain to be addressed in this work are rooted in the core methodology. In particular, we described how a mismatch event can trigger a systematic cross-layer adaptation process. However, in certain situations the execution of complex multilayer applications may generate more than one mismatch at (almost) the same time. While multiple mismatches can be addressed one at a time by our methodology, a metalevel coordinated selection of the sequence of templates to be applied may in some cases yield globally better adaptations. This is one of the directions for our future work. We also plan to deploy the methodology as an adaptation framework that allows third-party

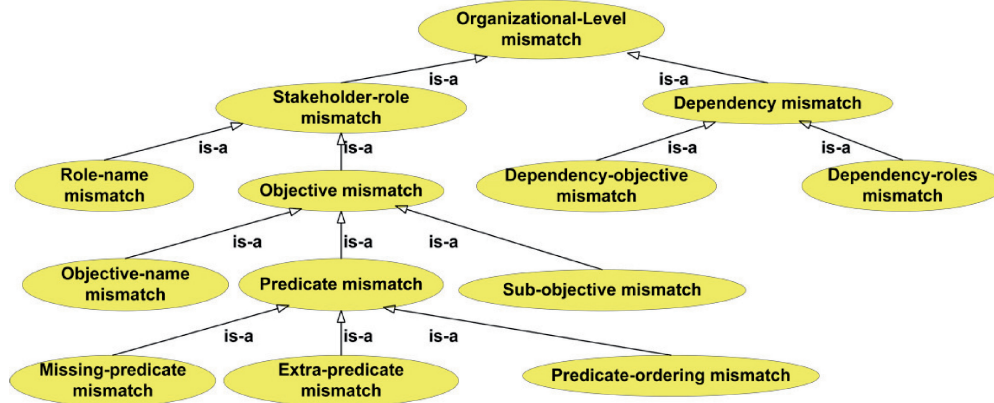


Fig. 9. A (partial) application mismatch taxonomy for the example organizational layer.

developers to easily define taxonomies of application mismatches and integrate adaptation logic into templates, as well as to match the templates needed to adapt multilayer applications. Another direction of future work is the formalization of criteria for checking the validity of templates. Finally, we intend to investigate an improved ranking system that takes into account the importance of templates through an analysis of user feedback and template link weights.

APPENDIX

1. INTRODUCTION

This is the appendix for Popescu [2011]. The main text describes a generic, formal methodology that supports flexible, cross-layer adaptation in multilayer applications. Semiformal metamodeling techniques are combined with more formal Petri nets, which are used to capture complex behavior. The main structural elements of the adaptation methodology are: templates (also known as patterns) that define generic adaptation solutions to common application mismatches, and taxonomies of application mismatches that provide classifications of common layer-specific application mismatches. An adaptation environment, modeled as a Petri net, captures the association of mismatches and templates. This appendix provides detail on these elements. Section 2 shows three sample taxonomies, one each for a three-layer system. Section 3 provides details of the algorithms used in the Petri net analysis of an adaptation environment containing mismatches and related templates. Section 4 illustrates a reachability graph for the crisis management scenario used in the article.

2. EXAMPLE ORGANIZATIONAL-LAYER, BEHAVIORAL-LAYER AND SERVICE-LAYER TAXONOMIES

Figure 9 describes part of a taxonomy of application mismatches for the organization application layer. An organizational layer provides a formalization of the application roles (stakeholders) and their objectives and relationships (dependencies) needed to support the achievement of the objectives. This taxonomy employs previously defined organizational concepts [ALIVE 2010]. For example, a *stakeholder-role mismatch* may be due to a *role-name mismatch* or *objective mismatch*. An *objective mismatch* may be due to, for example, a *predicate mismatch*, caused by a *missing-predicate mismatch* or an *extra-predicate mismatch* or a *predicate-ordering mismatch*.

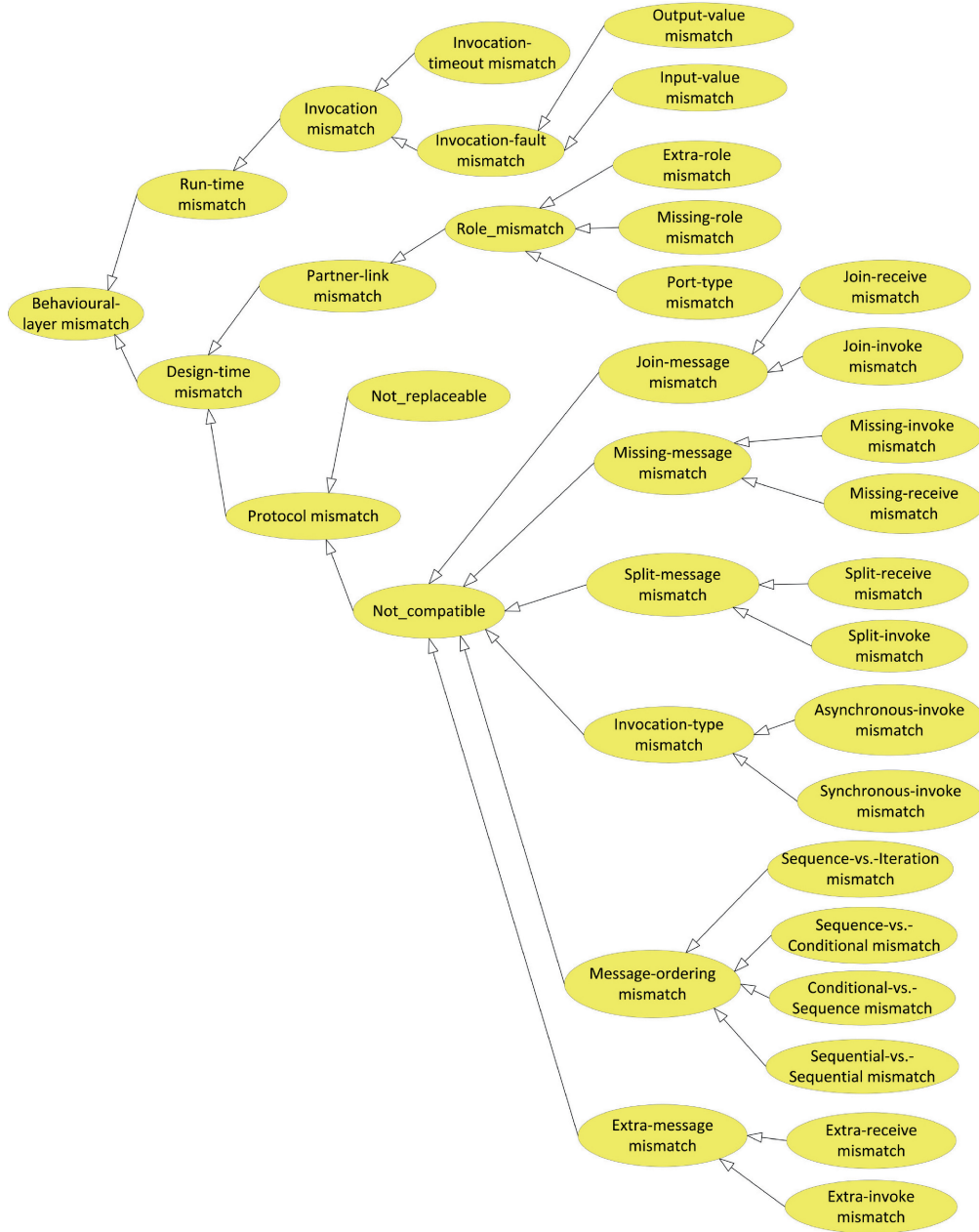


Fig. 10. A (partial) application mismatch taxonomy for the example behavioral layer.

Figure 10 illustrates part of the example taxonomy of application mismatches for a behavioral layer¹ of service-based applications. The partial taxonomy refines

¹In this context we refer to behavior as containing protocol information (i.e., orchestration of messages) and information about partner links (i.e., roles and port types as defined by the BPEL specification). Unless otherwise specified, the mismatches refer to the required protocol.

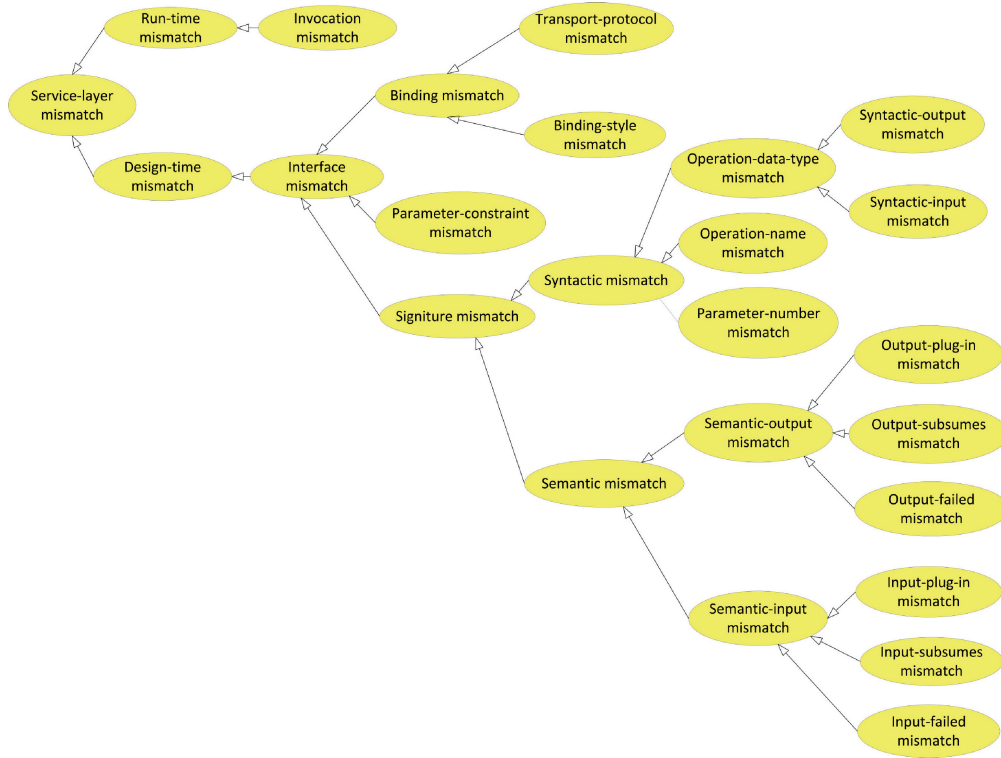


Fig. 11. A (partial) application mismatch taxonomy for the example service layer.

and extends previously defined behavioral mismatch patterns [Becker et al. 2004; Kongdenfha et al. 2009]. It refers to mismatches that may occur when comparing a *required* behavior specification with a *provided* one. We have split (design-time) protocol mismatches based on whether the required and provided protocols have to be *compatible* or *replaceable*. Protocol compatibility requires protocols to complement each other, for example, when one sends a message the other one has to receive it. Protocol replaceability requires the provided protocol to include the required one, that is, the provided protocol has to behave as the required one with respect to the clients of the required protocol. For example, when checking protocol compatibility, if both protocols define the same set of message exchanges (viz., invoke and receive operations) yet the required protocol executes these activities in a sequence while the provided protocol executes them in a loop, we then have a *sequential vs. iteration mismatch*. A *split-invoke mismatch* occurs when the required protocol sends a message (i.e., one invoke operation) yet the provided protocol expects to receive the same information as part of several messages (i.e., several receive operations).

Finally, Figure 11 presents part of a taxonomy of application mismatches for a service layer of a service-based application. The taxonomy refers to mismatches that may occur when comparing a *required* service specification with a *provided* one. An *interface mismatch* can be classified into *signature mismatch* (the required and provided interfaces have operations that differ either syntactically—different operation names, number, order, or type of input and output parameters, or semantically—use different ontological concepts for their inputs and outputs) and *parameter-constraint mismatch* (the required service interface imposes constraints—such as value range—on the

input or output parameters of one of its operations and they are different from what the provided interface defines). Similarly, an *operation data-type mismatch* can be classified into *syntactic-input mismatch* (a service operation has an input data type of unexpected/unknown type) and *syntactic-output mismatch* (a service operation has an output parameter of unexpected/unknown data type).

3. ALGORITHMS FOR THE ASSOCIATION AND DISSOCIATION OF TEMPLATES AND MISMATCHES

This section presents two algorithms that describe the steps needed for the update of adaptation environments when templates are added (i.e., bound to taxonomy mismatches) and respectively removed (i.e., unbound from taxonomy mismatches).

By construction of the PN, the following invariant property holds.

INV. For each mismatch m :

- (a) There is a token in $T(m)$ if and only if there exists a template associated with m .
- (b) There is a token in $UP(m)$ if and only if there is no template associated with m and there exists a template associated with some ancestor of m .
- (c) There is a token in $DOWN(m)$ if and only if there is no template associated with any ancestor of m and there exists a template associated with m or with a descendant of m .

The invariant can also be expressed as follows.

Corollary of INV. For each mismatch m there is at most one token in $T(p(m)) \cup UP(p(m)) \cup DOWN(m)$, where $p(m)$ denotes the parent of m in the taxonomy. The proof of the corollary is straightforward.

—By INV(b) and INV(a): token in $UP(p(m)) \Rightarrow$ no token in $T(p(m))$.

—By INV(c) and INV(a): token in $DOWN(m) \Rightarrow$ no token in $T(p(m))$.

—By INV(c) and INV(b): token in $DOWN(m) \Rightarrow$ no token in $UP(p(m))$.

Adaptation environments encode the adaptation logic of a multilayer application. The environment is set up at application design time by selecting taxonomies for (some of) the application layers and the templates bound to their mismatches. The following two subsections present the algorithms that define PN changes required by the association of templates to mismatches they can tackle, as well as their dissociation.

3.1. Associating Templates to Mismatches

The following ASSOCIATE algorithm updates the PN representing the adaptation environment when a template T is bound to a mismatch m . Intuitively, the algorithm updates (if needed) the representation of m 's ancestors so that adaptation events matching those mismatches can lead to selecting T as a more specific adaptation when no exact or more general adaptation is possible. Similarly, the algorithm updates (if needed) the representation of m 's descendants so that adaptation events matching those mismatches can lead to selecting T as a more general adaptation when no exact adaptation is possible.

The following proof establishes that INV is an invariant property for the ASSOCIATE algorithm.

PROOF. Initially INV trivially holds since there is no token present in $T(m)$, $UP(m)$, $DOWN(m)$ for all mismatches m . We now assume that INV holds and we show that it continues to hold after executing ASSOCIATE.

- (1) If $T(m)$ contains no token, then a token is put in $T(m)$ (line 3) to satisfy INV(a).
 - (1.1.) If there is a token in $UP(m)$ (line 4) then $UP(m)$ is the only place that must be updated (line 5). Indeed, by INV(b), there is a token associated with some

ALGORITHM ASSOCIATE:

```

1 IF no token in T(m)
2   THEN {
3     PUT token in T(m);
4     IF token in UP(m)
5       THEN REMOVE token from UP(m);
6     ELSE {
7       x = m;
8       WHILE x not root and no token in DOWN(x)
9         DO {
10           PUT token in DOWN(x);
11           x = p(x);
12         }
13       FOREACH child y of m DO Update(y);
14     }
15   }
16
17 Update(y) =
18 {
19   IF token in DOWN(y)
20     THEN REMOVE token from DOWN(y);
21   IF no token in T(y)
22     THEN {
23       PUT token in UP(y);
24       FOREACH child z of y DO Update(z);
25     }
26 }

```

ancestor of m and hence, by INV(b) and INV(a), all descendants d of m will continue to have a token either in $UP(d)$ or in $T(d)$. The same holds for the ancestors of m from $p(m)$ to $p^i(m)$, where $p^i(m)$ is the highest ancestor of m with an associated template. The remaining ancestors a from $p^{i+1}(m)$ to $p^{i+k}(m)$ (where $p^{i+k+1}(m)$ is the root) will instead continue to have a token in $DOWN(a)$.

- (1.2.) If instead there is no token in $UP(m)$ (line 6) then the descendants of m that are pointing to a more specific mismatch must be updated (line 16) to satisfy INV(c) and INV(b). Descendants are hence visited top-down, and for each descendant d of m , the token is moved from $DOWN(d)$ to $UP(d)$ (lines 22 and 25). The visit stops when a descendent with an associated template is encountered (line 23). Such a descendent c will continue to have a token in $T(c)$ (by INV(a)), while all the descendants f of c will continue to have a token in $UP(f)$ or in $T(f)$ by INV(b) and INV(a). To compute the update, we must also check whether there is a token in $DOWN(m)$ (line 7).

(1.2.1.) If there is a token in $DOWN(m)$ then, by INV(c), all ancestors a of m (but the root) already have a token in $DOWN(a)$ and they need not be updated.

(1.2.2.) If there is no token in $DOWN(m)$ (and no token in $UP(m)$), then by INV(c) (and by INV(b)), a token must be put in $DOWN(m)$. Moreover (lines 10–14) the parent p of m may have no token in $DOWN(p)$ (if none of the children c of p had a token in $DOWN(c)$), and in such a case a token must be now put in $DOWN(p)$. The other ancestors of m must be updated analogously until an ancestor a of m with a token in $DOWN(a)$ is (possibly) encountered.

- (2) If $T(m)$ already contains a token (line 1) then, by definition of INV, the PN does not need to be updated. $\square$

3.2. Dissociating Templates from Mismatches

The following DISASSOCIATE algorithm updates the PN representing the adaptation environment when a template T is unbound from a mismatch m . Intuitively, the algorithm updates (if needed) the representation of m 's ancestors so that adaptation events matching those mismatches can lead to selecting templates bound to descendants of m as more specific adaptations when no exact or more general adaptation is possible. Similarly, the algorithm updates (if needed) the representation of m 's descendants so that adaptation events matching those mismatches can lead to selecting templates bound to their descendants as more specific adaptations when no exact or more general adaptation is possible.

ALGORITHM DISSOCIATE:

```

1 IF T was the only template associated with m
2   THEN {
3     REMOVE token from T(m);
4     IF (m not root AND (token in T(p(m)) OR in UP(p(m)))
5       THEN PUT token in UP(m);
6     ELSE {
7       FOREACH child y of m DO Update2(y);
8       x=m;
9       WHILE (x not root AND token in DOWN(x)
10         AND no token in DOWN(z) for any child z of x)
11         DO {
12           REMOVE token from DOWN(x);
13           x = p(x);
14         }
15     }
16   }

17 Update2(y) =
18 {
19   IF no template in Tree(y)
20     THEN {
21       REMOVE token from UP(y);
22       FOREACH child z of y DO Update2(y);
23     }
24   ELSE {
25     PUT token in DOWN(y);
26     IF token in UP(y)
27       THEN {
28         REMOVE token from UP(y);
29         FOREACH child z of y DO Update2(y);
30       }
31   }
32 }
```

The following proof establishes that INV is an invariant property for the DISSOCIATE algorithm.

PROOF. We show that if INV holds on a given PN then it continues to hold after applying DISSOCIATE to such PN.

- (1) If T was the only template associated with m (line 1), then the token in $T(m)$ must be removed (line 3) to satisfy $\text{INV}(a)$ for m .
 - (1.1.) If (m is not the root and) there is a token in $T(p(m))$ or in $UP(m)$, then, by INV , $DOWN(m)$ contained no token and it must continue to do so. On the other hand, $UP(m)$ contained no token (by $\text{INV}(b)$, since $T(m)$ contained a token) but it must now contain a token (line 5). It is worth noting that no other update to the PN is needed since:
 - The ancestors a of m from $p(m)$ to $p^i(m)$, where $p^i(m)$ is the highest ancestor of m with an associated template, will continue to have a token either in $T(a)$ or in $UP(a)$ by $\text{INV}(a)$ and $\text{INV}(b)$. The remaining ancestors b from $p^{i+1}(m)$ to $p^{i+k}(m)$ (where $p^{i+k+1}(m)$ is the root) will instead continue to have a token in $DOWN(b)$.
 - By $\text{INV}(a)$ and $\text{INV}(b)$ the descendants d of m will continue to have a token either in $T(d)$ or in $UP(d)$.
 - (1.2.) If neither $T(p(m))$ nor $UP(p(m))$ contain a token (line 6) then, by $\text{INV}(a)$ and $\text{INV}(b)$, there is no ancestor of m with an associated template. For each child c of m :
 - (a) If $Tree(c)$ contains no templates (line 19) then by $\text{INV}(b)$, there is a token in $UP(c)$. To keep satisfying $\text{INV}(b)$, such token must be removed from $UP(c)$ (line 21) and the same must be done for all descendants of c (line 22).
 - (b) If $Tree(c)$ instead contains a template (line 24) then a token must be put in $DOWN(c)$ to satisfy $\text{INV}(c)$, and if there is a token in $UP(c)$ then it must be removed, and the same must be done for all descendants of c until a mismatch with an associated template is encountered. When such a descendant is encountered, a token must be placed in $DOWN(d)$ but the descendants f of d need not be updated as, by $\text{INV}(a)$ and $\text{INV}(b)$, they will continue to have a token either in $T(f)$ or in $UP(f)$.

If there is no token associated with any descendant of m , then the token in $DOWN(m)$ must be removed to satisfy $\text{INV}(c)$, and the same must be done for all ancestors of m (lines 8–15).
- (2) If T was not the only template associated with m then, by definition of INV , the PN does not need to be updated. $\square$

4. REACHABILITY GRAPH AND ANALYSIS FOR THE CRISIS MANAGEMENT SCENARIO

This section describes the reachability analysis of the adaptation environment Petri net. The matchmaker generates the possible sequences of templates that may tackle a raised event through a reachability analysis of the PN encoding the adaptation environment. The reachability graph of a Petri net represents all possible sequences of firing transitions given an initial configuration of tokens. The graph has PN markings as nodes and labeled arrows as arcs (see Figure 12). A marking defines a PN state that consists of the set of all places containing tokens. Each marking is represented in the figure as a set of 0's and 1's, which mark the absence and presence of tokens in the PN's places. The root of the graph is the initial marking, which shows the current distribution of tokens in the PN. New nodes in the graph are then obtained by considering all firing transitions given the token configuration in the initial marking. Each additional node is obtained by considering only one firing transition. An arrow linking two nodes states that the PN execution state evolves from one marking into another and it is labeled with the transition that produces the change. One may use tools such as WoPeD [WoPeD 2012] or Woflan [Verbeek and van der Aalst 2000] for the automated generation of reachability graphs from PNs.

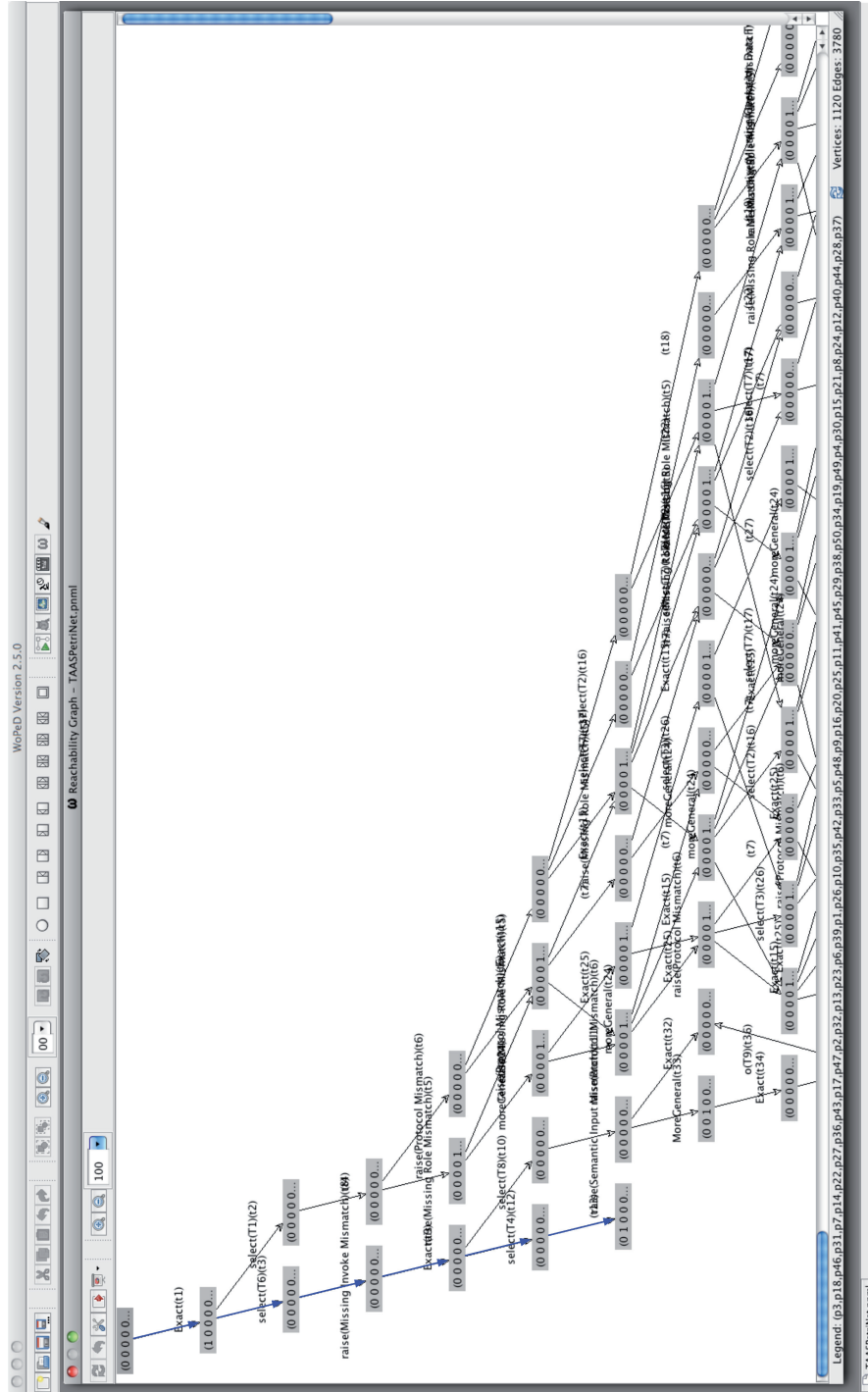


Fig. 12. Reachability graph corresponding to the partial adaptation environment in Figure 7.

Each path starting from the root node of the graph (having no incoming arcs) and ending at a node with no outgoing arcs denotes a possible sequence of templates. From each such path, the templates in the adaptation sequence correspond to the execution of the $invoke(Tx)$ and $select(Ty)$ transitions (see arc labels in the figure). The former indicates a direct template invocation, while the latter indicates template linkage through events. Note that the $invoke(Tx)$ transition is directly connected to $i(Tx)$ place in the adaptation environment, and hence $select(Tx)$ will not be executed in this case. Furthermore, successful execution paths from which an adaptation sequence can be synthesized correspond to reachability graph paths whose final markings may contain tokens in $T(m)$, $UP(m)$, $DOWN(m)$, and $o(Tx)$ places only. The main limitation of the reachability graph is that it has an infinite number of markings for unbounded PNs, that is PNs having at least one place that can contain an infinite number of tokens (due to loops in the PN). Karp and Miller [1969] proposed the Finite Reachability Tree (FRT) and its possible representation as a Coverability Graph (CG) as a solution to representing the infinite space-state of unbounded PNs. The key feature of the FRT is the introduction of the ω -symbol to represent a place with a potentially infinite number of tokens in markings resulting from some transitions' firing loops. The minimal CT was proposed by Finkel [1993], yet it is more computationally expensive. The FRT can be used to determine properties such as safeness, boundness, conservativeness, and coverability. Furthermore, it can be used to determine the liveness of the PN when the tree contains no ω -markings (i.e., a finite tree). However, the FRT cannot be used to determine liveness, deadlock, or reachability due to the loss of information caused by the ω -symbol. In order to tackle these properties, Wang et al. [2004] formalized the Modified Reachability Tree (MRT), which uses ω -numbers instead of ω -symbols. For example, a place in a marking to which it corresponds a $2\omega_1$ ω -number describes that the respective place holds an even number of tokens, not less than 2. The MRT can hence be used to tackle the extraction of adaptation sequences from unbounded PNs. While the algorithm for generating MRTs has the same order of complexity of the algorithm for generating FRTs, unfortunately, the reachability problem for PNs is known to be EXPSPACE-hard [Esparza and Nielsen 1994]. Note, however that, in our approach, the analysis of the adaptation environment is performed after the addition, removal, or modification of taxonomies and templates. The extraction of adaptation sequences takes place at query time. When adaptation environments are fairly static (i.e., are not updated very often), query results may be cached to achieve fast response times when adaptation needs arise.

REFERENCES

- ALIVE. 2010. Coordination, organisation and model driven approaches for dynamic, flexible, robust software and services engineering. European Commission 7th Framework Programme, ICT, <http://www.ist-alive.eu/>.
- BECKER, S., BROGI, A., GORTON, I., OVERHAGE, S., ROMANOVSKY, A., AND TIVOLI, M. 2004. Towards an engineering approach to component adaptation. In *Architecting Systems with Trustworthy Components*, 193–215.
- ESPARZA, J. AND NIELSEN, M. 1994. Decidability Issues for Petri Nets - A survey. *J. Inf. Process. Cybernet.* 30, 3, 143–160.
- FINKEL, A. 1993. The minimal coverability graph for Petri nets. In *Papers from the 12th International Conference on Applications and Theory of Petri Nets (APN'91)*, G. Rozenberg, Ed., Lecture Notes in Computer Science, vol. 674, Springer, 210–243.
- KARP, R. M. AND MILLER, R. E. 1969. Parallel program schemata. *J. Comput. Syst. Sci.* 3, 2, 147–195.
- KONGDENPHA, W., MOTAHARI-NEZHAD, H., BENATALLAH, B., CASATI, F., AND SAINT-PAUL, R. 2009. Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters. *IEEE Trans. Softw. Engin.* 2, 2, 94–107.
- POPESCU, R., STAIKOPOULOS, A., BROGI, A., LIU, P., AND CLARKE, S. 2011. A formalised, taxonomy-driven approach to cross-layer application adaptation. (submitted) *ACM Trans. Auton. Adapt. Syst.*

- VERBEEK, E. AND VAN DER AALST, W. M. P. 2000. Woflan 2.0: A petri-net-based workflow diagnosis tool. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN 00)*. Lecture Notes in Computer Science, vol. 1825, Springer, 475–484.
- WANG, F.-Y., GAO, Y., AND ZHOU, M. 2004. A modified reachability tree approach to analysis of unbounded petri nets. *IEEE Trans. System, Man Cybernet. B* 34, 1, 303–8.
- WoPeD. Workflow Petri net designer. <http://www.woped.org/>.

REFERENCES

- ALVES, A., ARKIN, A., ASKARY, S., BARRETO, C., BLOCH, B., ET AL. 2007. Web services business process execution language version 2.0. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>
- ALIVE. 2010. Coordination, organization and model-driven approaches for dynamic, flexible, robust software and services engineering. European Commission 7th Framework Programme, ICT. <http://www.ist-alive.eu/>
- AYERSANO, L., CANFORA, G., AND CIAMPI, A. 2004. An algorithm for Web service discovery through their composition. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*. IEEE Computer Society, Los Alamitos, 332–.
- BANSAL, S. AND VIDAL, J. M. 2003. Matchmaking of Web services based on the daml-s service model. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 926–927.
- BECKER, S., BROGI, A., GORTON, I., OVERHAGE, S., ROMANOVSKY, A., AND TIVOLI, M. 2004. Towards an engineering approach to component adaptation. In *Architecting Systems with Trustworthy Components*, 193–215.
- BENATALLAH, B., SAID HACID, M., LGER, A., REY, C., AND TOUMANI, F. 2005. On automating Web services discovery. *Vldb J.* 14, 84–96.
- BROGI, A. AND POPESCU, R. 2006. Automated generation of bpel adapters. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC'06)*. 27–39.
- CANAL, C., POIZAT, P., AND SALAUN, G. 2008. Model-Based adaptation of behavioral mismatching components. *IEEE Trans. Softw. Engin.* 34, 4, 546–563.
- CHARFI, A., DINKELAKER, T., AND MEZINI, M. 2009. A plug-in architecture for self-adaptive Web service compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS'09)*. 35–42.
- CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. 2001. Web services description language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>
- CORFINI, S. 2008. Composition-Oriented service discovery. Ph.D. thesis, University of Pisa.
- DIGNUM, V. 2003. A model for organizational interaction: Based on agents, founded on logic. Ph.D. thesis, Utrecht University.
- ERRADI, A., MAHESHWARI, P., AND TOSIC, V. 2006. Policy-Driven middleware for self-adaptation of Web services compositions. In *Proceedings of the 7th International Middleware Conference*. Lecture Notes in Computer Science, vol. 4290, Springer, 62–80.
- EZENWOYE, O. AND SADJADI, S. M. 2007. TRAP/BPEL: A framework for dynamic adaptation of composite services. In *Proceedings of the International Conference on Information Systems and Technologies (WEBIST'07)*.
- GERDS, C., MOOLJ, A. J., AND WOLF, K. 2008. Specifying and generating behavioral service adapter based on transformation rules. Tech. rep. CS-02-08, Universitat Rostock.
- GIØRVEN, E., ROUVOY, R., AND ELIASSEN, F. 2008. Cross-Layer self-adaptation of service-oriented architectures. In *Proceedings of the Workshop on Middleware for Service-Oriented Computing (MW4SOC'08)*. ACM Press, New York, 37–42.
- KARASTOYANOVA, D. AND LEYMAN, F. 2009. BPEL'n'Aspects: Adapting service orchestration logic. In *Proceedings of the International Conference on Web Services (ICWS'09)*. 222–229.
- KAZHAMIAKIN, R., PISTORE, M., AND ZENGIN, A. 2009. Cross-Layer adaptation and monitoring of service-based applications. In *Proceedings of the ICSOC/Service Wave Workshops*. A. Dan, F. Gittier, and F. Toumani, Eds., Lecture Notes in Computer Science, vol. 6275, Springer, 325–334.
- KLUSCH, M., FRIES, B., AND SYCARA, K. 2006. Automatic semantic Web service discovery with owls-mx. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*. ACM Press, New York, 915–922.
- KONGDENFHA, W., MOTAHARI-NEZHAD, H., BENATALLAH, B., CASATI, F., AND SAINT-PAUL, R. 2009. Mismatch patterns and adaptation aspects: A foundation for rapid development of Web service adapters. *IEEE Trans. Softw. Engin.* 2, 2, 94–107.

- LI, L. AND HORROCKS, I. 2003. A software framework for matchmaking based on semantic Web technology. In *Proceedings of the 12th International Conference on World Wide Web (WWW'03)*. ACM Press, New York, 331–339.
- MARTENS, A. 2005. Analyzing Web service based business processes. *Lecture Notes in Computer Science*, vol. 3442, Springer, 19–33.
- MCGUINNESS, D. AND VAN HARMELEN, F. 2004. OWL Web Ontology Language Overview. Web guide. <http://www.w3.org/TR/owl-features>
- MOKHTAR, S. B., GEORGANTAS, N., AND ISSARNY, V. 2005. Ad hoc composition of user tasks in pervasive computing environments. In *Software Composition*, 31–46.
- MOSER, O., ROSENBERG, F., AND DUSTDAR, S. 2008. Non-Intrusive monitoring and service adaptation for WS-BPEL. In *Proceedings of the International Conference on World Wide Web (WWW'08)*. ACM Press, New York, 815–824.
- MURATA, T. 1989. Petri nets: Properties, analysis and applications. *Proc. IEEE* 77, 4, 541–580.
- OMG. 2006. Meta object facility (MOF) version 2.0. <http://www.omg.org/spec/MOF/2.0/>
- OMG. 2009. Business process model and notation (BPMN) version 2. <http://www.omg.org/spec/BPMN/2.0/>
- PAGLIARECCI, F., PISTORE, M., SPALAZZI, L., AND TRAVERSO, P. 2007. Web service discovery at process-level based on semantic annotation. In *Sistemi Evoluti per Busi di Dati*. 220–232.
- PAOLUCCI, M., KAWAMURA, T., PAYNE, T. R., AND SYCARA, K. P. 2002. Semantic matching of Web services capabilities. In *Proceedings of the International Semantic Web Conference (ISWC'02)*. 333–347.
- PETRI, C. A. 1962. Kommunikation mit automaten. Ph.D. thesis, Institut für Instrumentelle Mathematik, Bonn.
- POPESCU, R., STAIKOPOULOS, A., AND CLARKE, S. 2009. An extensible monitoring and adaptation framework. In *Proceedings of the MONA+ ServiceWave Workshop on Monitoring, Adaptation and Beyond*.
- S-CUBE. 2011. Software services and systems network. <http://www.s-cube-network.eu>
- SRINIVASAN, N., PAOLUCCI, M., AND SYCARA, K. 2006. Semantic Web service discovery in the owls-s ide. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*. Vol. 6, IEEE Computer Society, Los Alamitos, CA, 109.2.
- SYU, J.-Y. 2004. An Ontology-Based Approach to Automatic Adaptation of Web Services. M.S. thesis, Department of Information Management, National Taiwan University.
- VAN DEN HEUVEL, W.-J., WEIGAND, H., AND HIEL, M. 2007. Configurable adapters: The substrate of self-adaptive Web services. In *Proceedings of the International Conference on Electronic Commerce (ICEC'07)*. 127–134.
- WILLIAM, J. H., LEE, W., AND NEWHOUSE, S. 2003. The iceni semantic service adaptation framework. In *Proceedings of the UK e-Science All Hands Meeting*. 17.
- ZHANG, J. AND CHENG, B. H. C. 2006. Model-Based development of dynamically adaptive software. In *Proceedings of the International Conference on Software Engineering (ICSE'06)*. ACM Press, New York, 371–380.

Received May 2010; revised November 2011; accepted December 2011