Robustness testing of composed real-time systems

Hacène Fouchal\textsuperscript{a,}\textsuperscript{*}, Antoine Rollet\textsuperscript{b} and Abbas Tarhini\textsuperscript{c}

\textsuperscript{a}CRESTIC, Université de Reims Champagne Ardenne, BP 1039 F-51687 Reims Cedex, France
\textsuperscript{b}LABRI/CNRS (UMR5800), Université de Bordeaux, 33405 Talence Cedex, France
\textsuperscript{c}Lebanese American University, Beirut, Lebanon

Abstract. In this paper, we suggest a methodology for testing robustness of Real-Time Component-Based Systems (RTCBS). A RTCBS system is described as a collection of components where each component is modeled as a Timed Input-Output Automaton (TIOA). For each component, we handle two specifications: a nominal one and a degraded one. We extract test sequences from the nominal specification and we inject automatically faults in order to model hostile environments. Then we present an adequate test architecture consisting of the System Under Test (SUT) of components, and a distributed tester that consists of a set of coordinating testers. Each tester is dedicated to test a single SUT component. A test execution algorithm is presented. Testing the SUT is divided into two phases. In the first phase, the tester executes the generated test sequences of each component in isolation and records the feedback of this experimentation. The robustness is checked by verifying if the recorded results are accepted by the degraded specification of each component. If all components are robust according to the inserted hazards, we check the robustness of communications between components respecting the same process described before.

Keywords: Real-Time system, timed automata, component based system, validation, testing, robustness

1. Introduction

Recently, software is urged to be more complex consisting of many independent distributed components running concurrently on heterogeneous networks; consequently, facilitating the creation of emerging technologies such as commercial Off-The-Shelf (COTS) products which are becoming a market reality [15].

Traditional testing methods that ordinary COTS products undergo are not thorough enough to guarantee reliability; still, many of these products are integrated temporally in critical real-time component-based systems. Such integration may lead to architectural mismatches when assembling components with incorrect behavior [7], leaving the system in a hostile environment. The criticality of such systems requires the creation of software components that can function correctly even when faced with improper usage or stressful environmental conditions. The degree of tolerance to such situations is referred to as a component’s robustness.

This paper deals with robustness testing for Real-Time Component-Based Systems (RCBS). Our methodology consists of (1) representing each of the SUT components as a Timed Input-Output Automata (TIOA), (2) generating and mutating test sequences from each TIOA, (3) executing mutated test sequences

\textsuperscript{*}Corresponding author. E-mail: Hacene.Fouchal@univ-reims.fr.
using a distributed tester that consists of a set of coordinating testers. Each tester is dedicated to test a single SUT component.

The mutation method inserts hazards to test sequences generated from the component’s nominal (original) specification. The mutated test sequences are used to test the system’s robustness according to a corresponding degraded specification which describes the minimal authorized behavior in case of unexpected situations. A system is robust if it is able to operate correctly in the presence of invalid inputs or stressful environment (IEEE definition). These timed test sequences are divided into two sets. The first set is used to check the robustness of each component in isolation, and the second set is used to test the correctness of communication between the components integrated in a RTCBS; and thus the robustness of the whole system. Then, we focus on the execution of these timed test sequences. The execution of timed-mutated test sequences on the SUT is divided into two phases. In the first phase, the tester executes a set of test sequences to test the robustness of each component in isolation, and register the feedback from the components into traces. Next after checking the registered traces with the corresponding degraded specification, the tester produces a verdict success or failure relative to the inserted hazards. If success, the components are robust—enough stand-alone objects. Otherwise, components are identified as not robust. In the second phase, the tester also executes and monitor timed test sequences, but to test the robustness of interaction between components integrated in a RTCBS and thus testing the system’s robustness. In both steps, our main concern is to monitor the behavior and cover the entire timing interval where an action has to be submitted to the SUT; thus the action of a transition will be submitted as soon as we enter the interval and at the ending instant of that interval.

In the following, we present our proposed approach in details. Section 2 provides a review on related work done on testing distributed and component based systems. Section 3 provides a background of different modeling techniques used to represent CBS. In Section 4 we focus on our method for testing Real-Time Component Based Systems. The timed test sequence mutation method is presented. It explains how we insert hazards; next, we detail the acceptance relation used in robustness testing. In Section 5, we detail the testing architecture as well as the test execution process. Finally, we conclude in Section 6.

2. Related work

Distributed systems may be viewed as a set of components connected via a communication system, where each component sends and/or receives messages through one or more communication channel. Distributed testing was studied with different test architectures [10] proposed a distributed tester, with the distribution of global test sequence (GTS) into local test sequences executed over local testers, where the synchronization and the fault detection problems are discussed.

In Component Based Systems (CBS), components must interact with different external elements [17]. Those components may run on single, multiple, distributed or network systems in a sequential or individual concurrent manner, and may communicate with every other component. Such systems require a tight and loose coupling of components of diverse granularity. [10] handled testing distributed real-time systems, and proposed a method for solving the controllability and observability problem by holding some timing constraints.

Robustness testing has been studied in many areas. It is based on insertion of hazards into test sequences. In the following, we give an overview of selected works on robustness testing and fault injection methods. Some studies are based on fault injection by using special-purpose hardware which causes electric disturbances. [9] applies heavy-ion radiations on replicated CPU boards to measure
radiation effects on a system. Another fault injection approach is developed, using software. It is called Software Implemented Fault Injection (SWIFI). Among these studies is the FIAT system that modifies the binary image of a process in memory (fault injection at compile time) [2,16] proposed FTAPE that combines system-wide fault injection with a controllable workload in order to create high stress conditions and fault propagation for the machine. [11] proposed an object oriented approach to test software robustness-based on parameter data types rather than component functionality. It is implemented in the Ballista tool. It is a black box approach, i.e. the tester uses only the description of the component interface in terms of parameters and data types. They consider that most robustness failures of software are caused by code that simply neglects to test for invalid inputs. The simulation approach has been used to measure the fault tolerance. It has been adopted in several tools, such that DEPEND [8]. Some other studies tried to mix these different approaches.

Robustness testing suggests mainly to consider all possible faults that a system can execute. The designer has to insert them into the specification and finally derives test sequences from the modified specification. In the following section we discuss how a real-time component-based system is best modeled.

Another approach consists using model-based test generation. The main difficulty of such methods is to describe the hazards in the model. Many works consider such an approach, see for example [4, 13,14]. In [14], the authors propose an approach based on an increased specification used to specify the acceptable behaviors in presence of hazards. In [4], Fernandez et al. use a formal fault model in order to build a “mutant” specification. They use a fault model in order to add “fault” transitions in the specification; they define a robustness relation based on a robustness property. Contrary to our approach, they do not permit to integrate unexpected inputs in the model.

3. Modeling

Modeling component-based systems requires us to recall the definition of components that are the building blocks of such system. Szyperski [15] defined components as: a component is a unit of composition with contractually specified interfaces and fully explicit context dependencies that can be deployed independently and is subject to third-party composition.

Software architecture (SA) dynamics that represent the individual components and their interactions are used in building component-based systems. Brinksma [3] distinguished between the following perspectives of CBS: (1) the individual component implementation: the executable realization of a component, (2) the component interface (interaction): summarizes the properties of the component that are externally visible to the other parts of the system, and (3) real-time component: component technology for real-time systems should support specification and prediction of timing.

In fact, the time factor problem in reactive real-time component based systems (RTCBS) is not enough investigated [3]. The time issue should be included in the model of such systems, where an action from the environment is modeled as an input event, while the reaction from the system is an output event. A Timed Input Output Automata (TIOA) is able to specify a sequence of inputs and their corresponding outputs; moreover, it shows timing constraints on events occurring in the system. Consequently, a TIOA best represents the (SA) dynamics of a TCBS.

In our model, we use continuous time. This is due to the fact that is more realistic. A solid theoretical foundation of timed automata (TA) is defined by Alur-Dill [1] as a finite set of states and a finite set of clocks which are real-valued variables. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started or reset. Each transition of the system might reset some
of the clocks, and has an associated enabling condition (EC) which is a constraint on the values of the clocks. A transition is enabled only if its corresponding state is reached and the current clock values satisfy its enabling condition.

In this study, we choose to model the communication of the whole system and the individual components of the RTCBS as TIOA. In the next section, we first present a formal definition of the TIOA that represents a single component in a SUT; next, we describe the test sequence mutation method.

4. A Robustness testing methodology for RTCBS

The purpose of our testing is to check how the system reacts to hazards, and consequently to come stressful situations. A hazard could be defined as any unexpected event from the environment. In our study we handle external hazards. External hazards are modeled as actions received by the system or erroneous timing constraints. The automatic generation and integration of hazards in the test sequences can be respectively found in [5,13].

4.1. Single component representation

Each component of the RTCBS is described by a nominal and a degraded specification. A nominal specification describes the normal behavior of the system. A degraded specification describes the vital functionalities and the minimal required behavior of the system. Each of these specifications is represented by input-complete, reduced, strongly connected timed input-output automata TIOA. A TIOA is defined formally as follows:

**Definition 1** (Timed Input Output Automaton (TIOA)). A TIOA is defined by \( M = (S, A, C, T, s_0) \) where \( S \) is a finite set of states, \( s_0 \) is the initial state, and \( A \) is a set of actions. \( A \) is partitioned into two sets: \( A_I \) is the set of input actions (written \(?i\)), \( A_O \) is the set of output actions (written \(!o\)). Input and output actions are referred to as visible actions. \( C \) is a set of clocks. \( T \) is a transition set having the form \( \{Tr_1, Tr_2 \ldots Tr_n\} \). \( Tr_i = < s; a; d; EC; C_s > \), where \( s \) and \( d \) are starting and destination states. ‘\( a \)’ is the action of the transition. EC is an enabling condition evaluate to the result of the formula \( a \sim b \) where \( \sim \in \{\prec, \succ, \preceq, \succeq, =\} \) or to a constant valued either true or false. \( C_s \) is a set of clocks to be reset at the execution of a transition. Whenever enabled, if \( Tr_i \) is output transition, it should be directly executed reaching state ‘\( d \)’ without staying indefinitely in state \( s \). The initial state of TIOA is \( s_0 \). \( s_0 \) can execute only input transitions. After the execution of \( Tr_i \) all clocks in \( C_s \) are reset.

Below we introduce some definitions and notations needed in our methodology.

**Definition 2** (input-complete). A TIOA is said to be input-complete if all states accept any input \( a \in A_I \) at any instant.

**Definition 3** (minimal). A TIOA is minimal if for every pair of states \( s_i, s_j \) belong to TIOA, there is no sequence of events \( s \) when applied to both states, TIOA reaches the same state \( s_r \).

**Definition 4** (strongly connected). A TIOA is strongly connected if for every pair of states \( s_i, s_j \) belong to TIOA; there exists a sequence of events \( x \) such that state \( s_j \) is reachable from \( s_i \) if \( x \) is applied.
Definition 5 (Completely specified). A TIOA $M$ is said to be completely specified if all states of $M$ accept any action $a \in A$.

Definition 6 (Controllable state). A state $s \in S$ of a TIOA $M$ is called a controllable state, denoted by $\delta(s)$ if there is no other possibility than applying an input action to reach another state of $M$, i.e.: if there exists a state $s' \in S - \{s\}$ and an edge $e = (s, a, s', \lambda, \delta)$ then $a \in \{A_I\}$.

Notations

$P_m$ is a subcomponent indexed $m$. $S_{Pm}$ is a TIOA nominal specification that specifies a component $P_m$ of the system under test (SUT). SUT is composed of a set of components $P_1, P_2, \ldots, P_N$. $S_{Cm}$ is a TIOA communication nominal specification that specifies the communication of the whole system. $S_{Pm}^{d}$ is a degraded specification for component $P_m$. $S_{Cm}^{d}$ is a TIOA degraded specification for the communication of the whole system.

Definition 7 (Order relation ($T(\prec_T)$)). We define the order relation on two actions $a_i, a_j$ of a path $T$, denoted $a_i \prec_T a_j$, meaning that action $a_i$ occurs before action $a_j$ in $T$.

Definition 8 ($\subset_D$). Let $M = (S, A, C, T, s_0)$ and $M' = (S', A', C', T', s'_0)$ be two TIOA, $\text{Tr}(M)$ and $\text{Tr}(M')$ respectively all traces of $M$ from $s_0$ and $M'$ from $s'_0$. $M' \subset_D M$ iff $s_0 = s'_0$ and $A' \subset A$ and $\forall t' \in \text{Tr}(M') \exists t \in \text{Tr}(M)$ such that $(a, b) \in (A')^2$, $a \prec t' b \implies a \prec t b$.

Figure 1 shows an example of a nominal specification TIOA $M$ with initial state $s_1$. A transition is represented by an arrow between two states and labeled by $(\text{action}; \text{EC}; C_a)$. The set of actions $A = \{?\text{temperatureReq}, !\text{temperature}, ?\text{posReq}, !\text{pos}, ?\text{moveMode}, ?\text{endMoveMode}\}$; and the set of states $S = \{s_1, s_2, s_3, s_4\}$.

Assumptions

For each component, we consider the following assumptions:
Degraded specifications $S_{Pm}^d$ and corresponding nominal specifications $S_{Pm}^n$, respect the relation $S_{Pm}^d \subset D S_{Pm}^n$.

- Test sequences should be extracted from paths in nominal specification ending in a controllable state,
- Every output transition must have a time bound,
- Clocks of the nominal and the degraded specifications are completely independent,
- Specifications are supposed to be deterministic and strongly connected,
- For all specifications, each state have outgoing transitions that are either all inputs, or all outputs.

4.2. Test Sequence mutation method

4.2.1. Test Sequence Generation

In this work, we are not concerned with the test sequence generation. In our work, we use timed sequences, using the following definition:

**Definition 9** (Timed test sequence, input triplets). A timed test sequence extracted from a TIOA $M = (S, A, C, T, s_0)$ is a set of ordered triplets $Tr_i = (a_i, EC_i, C_{si})$, $1 \leq i \leq n$ such that $a_i \in A_i$, $EC_i$ is an enabling condition on $C$ and $C_{si}$ is a set of clocks of $C$ to reset. We denote $T = \{Tr_1, Tr_2, \ldots, Tr_n\}$ such a sequence. The set of such kinds of triplets $Tr_k$ is denoted TT.

We assume timed test sequences are already generated from the nominal specification using the test purpose technique [6]. In [6], the generated test sequences start with an input event and end with output events. For each controllable state we derive a sequence able to characterize this state among all others. The generated test sequences contain events that may not be included in the degraded specification, such events are identified as non-relevant events, other events are identified as relevant events. The following notation is added:

4.2.2. Notation (input triplets, non-relevant input triplets)

We define the set of input triplets, $TT_{in}$, a subset of in which the action is an input action (i.e. an element of $A_I$). We also define the set of non-relevant input triplets, $TT_{nrel}$, a subset of in which the action is not a relevant input action (i.e. the action is an element of $A_I - A'_I$). And finally, we define $TT_{nrelb}$ a subset of $TT_{nrel}$ in which all the triplets have a bounded enabling condition.

Next, we present our method for mutating those test sequences by insertion of input hazards.

4.2.3. Hazards

Hazards are inserted in the generated test sequences to simulate a hostile environment. In fact, we modify only the inputs in the testing sequences since the outputs are generated by the system (they are external to the system). Besides, because we check if the system is able to react well in presence of relevant actions, we do not allow to modify any input action that belongs to the set of input actions of the degraded specification.

Suppose that $M = (S, A, C, T, s_0)$ is the nominal specification, and $M' = (S', A', C', T', s'_0)$ is the degraded one. We choose a “pragmatic” approach for this integration. The designer decides which scenario he wants to integrate in a sequence based on previous experience. Let $T = \{Tr_1, Tr_2, \ldots, Tr_n\}$ this timed test sequence, extracted from $M$. We propose five possibilities:

1. Replacing an input action.
   In this item, we simulate the fact that another component sends an unexpected action to the tested component. The designer chooses an input transition $Tr_i$ which is not relevant, then we change this action with another one, still not relevant.
4.2.4 Integration function

Let \( f_{ij} : \mathbb{TT}^n \times C \rightarrow \mathbb{TT}^{n-1} \cup \mathbb{TT}^{n+1} \) be defined as:

\( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \ldots, \mathcal{T}_{j-1}, \mathcal{T}_j, \ldots, \mathcal{T}_n, d) \rightarrow \) if \( d = \text{Replace} \) and \( \mathcal{T}_i \in \mathbb{TT}_{nrel} \) cf 1.

else if \( d = \text{Time} \) and \( \mathcal{T}_i \in \mathbb{TT}_{nrel} \) cf 2.

else if \( d = \text{Exchange} \) and \( \mathcal{T}_i \in \mathbb{TT}_{nrel} \) and \( \mathcal{T}_j \in \mathbb{TT}_{nrel} \) cf 3.

else if \( d = \text{Add} \) and \( \mathcal{T}_i \in \mathbb{TT}_{in} \) cf 4.

else if \( d = \text{Remove} \) and \( \mathcal{T}_i \in \mathbb{TT}_{nrel} \) cf 5.

else \( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \ldots, \mathcal{T}_{j-1}, \mathcal{T}_j, \ldots, \mathcal{T}_n) \)

1. \( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, (a, \mathcal{EC}_i, \mathcal{C}_i), \mathcal{T}_{i+1}, \ldots, \mathcal{T}_j, \ldots, \mathcal{T}_n) \)
2. \( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, (a_i, \text{compl}(\mathcal{EC}_i), \mathcal{C}_i), \mathcal{T}_{i+1}, \ldots, \mathcal{T}_j, \ldots, \mathcal{T}_n) \)
3. \( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_j, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_{j-1}, \mathcal{T}_j, \mathcal{T}_{j+1}, \ldots, \mathcal{T}_n) \)
4. \( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, (b, t, \mathcal{EC}_i'), \mathcal{T}_i, \ldots, \mathcal{T}_{j-1}, \mathcal{T}_j, \ldots, \mathcal{T}_n) \) with \( \mathcal{EC}_i' \subset \mathcal{EC}_i \)
5. \( (\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_j, \ldots, \mathcal{T}_n) \)

2. Changing the instant of an input action occurrence.

Here, we simulate the fact that another component sends the expected action, but not at the right moment. This could happen for example in case of heavy CPU processing. The designer chooses a transition \( \mathcal{T}_i \) of \( \mathcal{T} \) with a bounded clock constraint. Then we change this constraint with another enabling condition, disjoint from the original one. In practice, we just delay the occurrence of the transition with an amount \( \varepsilon \), such that the occurrence arrives later than expected.

3. Exchanging two input actions.

In this item we simulate other components having scheduling problems. The designer chooses two non-relevant transitions \( \mathcal{T}_i \) and \( \mathcal{T}_j \). Then, we permute these two actions in the resulting sequence.

4. Adding an unexpected transition.

Here we simulate the fact that a component of the whole system sends an unexpected supplementary action. This may be caused for example by some troubles with a sensor. The designer chooses a transition \( \mathcal{T}_i \) of \( \mathcal{T} \) and a non-relevant action \( b \). Then we add a transition in the sequence with the input action \( b \) without changing the timing of the whole sequence.

5. Removing a transition.

In this part we simulate the fact that information is lost in the system. This could happen for many reasons, such that a problem in the communication channel. The designer chooses a non relevant transition \( \mathcal{T}_i \) of \( \mathcal{T} \) and we just remove it from the sequence.

4.2.4 Integration function

It is possible to formalize it by defining an integration function \( f_{ij} \) applied on a sequence. The aim of this function is to formalize the integration of hazards in a timed sequence. In order to write this function, some elements are needed. We define:

- A set \( C = \{ \text{Replace}, \text{Time}, \text{Exchange}, \text{Add}, \text{Remove} \} \) corresponding to the five possibilities given above,
- The three elements of a triplet \( \mathcal{T}_i \) of the origin set of the function are also written \( (a_i, \mathcal{EC}_i, \mathcal{C}_i) \),
- \( a' \) and \( b' \) are two randomly chosen elements of \( A_i - A_i' \),
- Let \( \mathcal{EC} \) an enabling condition, we define the function \( \text{compl}(\mathcal{EC}) \) which returns the complementary of \( \mathcal{EC} \) in the set of clock values

Let \( i \) and \( j \) two integers such that \( 0 < i < j < n \) (chosen by the designer), the integration function is defined as:
4.3. Mutated Sequence Acceptance – theoretical framework

In this section we illustrate the acceptance of the execution of a mutated test sequence by a degraded specification using the accept$_R$ relation. The aim of the Acceptance Relation accept$_R$ is to validate only the relevant events in a mutated test sequence with respect to the degraded specification.

We assume here that test sequences have been executed on SUT components, and the feedback results are recorded into execution traces also denoted timed word $T(\text{Seq})$. Based on Alur-Dill work [1], the formal definition of timed words is as follows:

**Definition 10** (Time sequence) A time sequence $\tau = \tau_1\tau_2\ldots$ is an infinite sequence of time values $\tau_i \in \mathbb{R}$ with $\tau_i > 0$, satisfying the following constraints:

- Monotonicity: $\tau$ increases strictly monotonically; i.e., $\tau_i < \tau_{i+1}$ for all $i \geq 1$.
- Progress: for every $t \in \mathbb{R}$ there is some $i \geq 1$ such that $\tau_i > t$.

**Definition 11** (Timed word) A timed word over an alphabet $A$ is a pair $(\sigma, \tau)$ where $\sigma = \sigma_1\sigma_2\ldots$ is an infinite word over $A$ and $\tau$ is a time sequence. An example of possible execution traces set is given in Fig. 2.

The aim of this part is to give the rules to decide whether an execution trace is accepted by the degraded specification. Figure 3 shows an example of a degraded specification. If all execution traces are accepted, then the component implementation is considered as robust enough (according to the hazards inserted in test sequences). To express this relation, we will use the notion of a run defined by Alur and Dill in [1], that gives a relation between a timed word and a timed automaton of the degraded specification. This relation is called robustness acceptance.

We recall some definitions [1] required to introduce our relation.
Definition 12 (Clock interpretation). A clock interpretation \( v \) for a set \( C \) of clocks assigns a real value to each clock; that is, it is mapping from \( C \) to \( \mathbb{R} \). We say that a clock interpretation \( v \) for \( C \) satisfies a clock constraint \( \text{EC} \) over \( C \) iff \( \text{EC} \) evaluates to true using the values given by \( v \).

Definition 13 (Run). A run \( r \), denoted by \( (\sigma, \tau) \) of a TIOA \( M = (S, A, C, T, s_0) \) over a timed word \( (\sigma, \tau) \) is a sequence of the form: \( R: <s_0, v_0> [\sigma_1 \rightarrow \tau_1] <s_1, v_1> [\sigma_2 \rightarrow \tau_2] <s_0, v_0> [\sigma_1 \rightarrow \tau_1] \ldots \) where \( s_i \in A \) and \( v_i \in [C \rightarrow \mathbb{R}] \), for all \( i \geq 0 \), satisfying the following requirements:

- **Initiation**: \( v_0(x) = 0 \) for all \( x \in C \).
- **Consecution**: for all \( i \geq 1 \), there is an edge in \( T \) of the form \( <s_{i-1}, \tau_i, s_i, \text{EC}_i, C_{si}> \) such that \( (\nu_{i-1} + \tau_i - \tau_{i-1}) \) satisfies \( \text{EC}_i \) and \( \nu_i \) equals \( [C_{si} - 0] (\nu_{i-1} + \tau_i - \tau_{i-1}) \).

For any test sequence \( T s_m \) (obtained from the nominal specification \( M \)), we may not be able to execute a run of \( M' \) over a timed word obtained from the test execution. The reason is that these words contain some actions not expected by the degraded specification \( M' \). As a consequence, no run would be executed in all cases.

For example, suppose that we would like to find a run of the degraded specification \( M' \) of Fig. 3 over the first timed word obtained from \( M = (S, A, C, T, s_0) \) in Fig. 2, we notice that it is not possible because the last action of the word (\( \text{moveMode} \)) is not in \( M' \). To avoid this situation, we propose to complete the TIOA \( M' \) by adding to each state transition loops labeled by (1) all complementary actions (input/output) of the alphabet and (2) other loops labeled with the same action however, with complementary time constraint. We further assume that the implementation of a component is also input-complete. Then any run of the completed \( M' (M'_{\text{compl}}) \) is possible over any timed word \( w \) based on the restriction that \( w \) and \( M'_{\text{compl}} \) use the same alphabet and that the time constraints are compatible.

The idea of robustness acceptance is that the tester sends an input (or several inputs), and waits for an answer (or more) respecting time constraints bounded in time. Then we have two cases:

- if the input was not expected in the degraded specification \( M' \), it means that it was not a relevant action. Thus, during the run of \( M'_{\text{compl}} \), the system stays in the same state, waiting for a relevant input.
- if the input was expected in the degraded specification \( M' \), the run of \( M'_{\text{compl}} \) passes the transitions.

Then the only way to come back in a controllable state is that the implementation of the component gives the output(s) expected in \( M' \). If not, the run is blocked in a non controllable state.

Finally, if the run of \( M'_{\text{compl}} \) ends on a controllable state, then the trace (in fact the timed run) is accepted in the sense of robustness. So we define the following relation:

Definition 14 (\text{accept}_{R}'). A finite word \( w = (\sigma, \tau) \) is accepted in the sense of robustness by a TIOA specification \( M' \), denoted \( M'_{\text{accept}_{R}w} \) iff there exists a run \( (\sigma, \tau) \) of \( M'_{\text{compl}} \) over \( w \) such that the last state of \( M'_{\text{compl}} \) reached by \( r \) is controllable. By extension, for a set of timed words \( \gamma \), we denote \( M'_{\text{accept}_{R}\gamma} \) iff \( \forall w \in \gamma, M'_{\text{accept}_{R}w} \).

5. Test Architecture and execution

In this work, we consider a real-time component-based system that consists of several components each of which is able to interact with the environment. Figure 4 illustrates our test architecture. It consists of a set of distributed local testers. For each component, \( P_m \), of the system, a dedicated Tester,
Fig. 4. Test Architecture for the whole system.

Tm, is assigned. This part is inspired by the work done in [12] dedicated to an adapted test architecture for timed systems. Below we present the notations needed in this section. In the following subsections, we describe the scheme for component’s communication; next, we present the architecture of each local tester, and finally we explain the test execution process.

Notations

Tm is a tester for a component m. N is the number of local testers, and thus the number of components. OTi is an output from a component Pm sent to tester Ti (tester for component Pi) through tester Tm.

LTSm = {LTm_S1, LTm_S2, ..., LTm_Sn} is a set of local mutated test sequences for a component Pm. Each LTm_Si has the form Trm_i.Trm_i ... Trm_n. CTs is an LTSm with at least an event Trm_k requiring a component Pm to get input from (resp. send output to) another component Pi.

ITs is an interoperable mutated test sequence set that contains all CTs. Ts is a set of LTS and CTs. For simplicity we refer to LTS and CTs as Ts.

ioQ = {ioQ1, ioQ2, ..., ioQN} where each ioQm is an input/output priority queue dedicated for tester Tm. ioQm holds inputs sent by any tester Ti to component Pm. Trm_j is a transition indexed j and is an input from tester Tm.

wait(Tm), signal(Tm): wait and signal are two atomic functions. Wait(Tm) will pause the execution of tester Tm. Signal(Tm) will resume the execution of tester Tm.

Components communicate with each other through their corresponding testers. That is, an output from component Pi to component Pj is done by sending the output to tester Ti which in turns sends to tester Tj, next, Tj forwards the output to Pj. It is important to note that in our model the time taken by the communication between components through the testers is considered with accuracy.

5.1. Local tester representation

Local testers execute local test sequences (Ts) on their corresponding components (P) by sending input actions and receiving output actions in the form OTi (i = {1, ..., m, ..., N}). Figure 5 details the units of each local tester which consists of the following: a test executor unit (TEU), a test monitor unit (TMU), a local input/output queue (ioQ) and a local clock. The job of the test executor unit (TEU)
is to apply mutated-timed test sequences to a component, and the job of the test monitor unit (TMU) is to register the feedback from the components into execution traces, then validate them with the degraded specification. Embedding the clocks in testers, enables the tester to tell about the time an event occurred instantaneously with no communication delay. Moreover, each tester’s clock is assumed synchronized with its corresponding component’s clock. The assumption of synchronization between a tester’s clock and the corresponding component’s clock is essential; otherwise, we will not be able to tell about the clock of the black-box component.

5.2. Tester coordination

Testers communicate with each other through the input/output queues (ioQ). In their communication, the following testers’ communication scheme is respected. The output transition from tester $T_i$, sent as an input, to tester $T_j$ will wait on queue $ioQ_j$ until this transition is enabled in $T_j$. On the other hand, the execution of a communication test sequence $CTs_j$ by tester $T_j$ will pause if this test sequence requires $P_j$ to wait on an input from component $P_i$. The execution in tester $T_j$ will resume after receiving the needed input from tester $T_i$; and thus, testers give a higher priority to handle inputs received from the components over inputs received from other testers that are stored in local testers’ queues. Testing any event on an interval is done at two instants: at the entering instant and at the ending instant of that interval. We assume that such execution covers the entire interval. During the test execution process, temporal and behavioral properties of the SUT are checked by applying timed test sequences. Such sequences contains sending and reception of actions. Below we define the Time rule for covering the time space:

Let “$A$” be an input action to the SUT from the tester. The tester sends “$A$” as soon as the enabling condition of $A$ satisfies the instant “$t$” reached by the local tester’s clock, and another experiment is performed at the latest instant satisfying $EC$. ($t \in EC$).

5.3. Test execution

The time-test execution scheme is defined as follows: A time-test execution will be declared as success iff the execution of all relevant events respecting the respects the accept$R$ timing conditions of their transitions specified in the degraded system. Otherwise, it will be declared as failure. The testing execution process is done in two phases. The first phase tests the robustness of each component
separately, and the second phase tests the robustness of communication among components. In both phases, each tester $T_m$ executes, using TEU, the corresponding mutated test sequences and records, using TMU, the corresponding feedbacks from each component into execution traces without instantaneous evaluation. Next, based on the accept$_R$ relation, we check the robustness of each recorded execution trace from each component with its corresponding degraded specification. In the first phase, we ignore all communication requests from other components; and thus, all inputs from testers are sent at the instant those inputs are needed based on the information in the nominal specification without considering the communication from any other tester $T_i$, and therefore, here we are checking the robustness of the component in isolation, without taking any communication input from other components.

An execution of a mutated test sequence $T_{Sm}$ by $T_m$ gives a verdict success, iff, in its corresponding recorded execution trace, the reception of outputs from $P_m$ are accepted in the degraded specification $S_{Pm}^d$ and the timing constraints respect the time-test execution scheme defined above. Only communication test sequences $CT_{Sm}$ are added to the set $IT_{Sm}$ to be re-executed in the second phase. At any failure, the component and the failing test sequence are identified.

In the second phase, each tester $T_m$ re-executes, using TEU, only its corresponding communication test sequence $IT_{Sm}$ generated from phase 1. Testers communicate with each other via input/output queues respecting the scheme of testers’ coordination mentioned above. The execution of tester $T_m$ to a communication test sequence $IT_{Sm}$ gives a verdict success, iff, in its corresponding recorded execution trace, the sending inputs and reception of outputs $O_{Ti}^m$ from $P_m$ or $T_i$ are accepted in the degraded specification $S_{Pm}^d$ respecting the timing constraints stated by the time-test execution scheme. For example, when the output $O_{Ti}^m$ is sent as an input to tester $T_i$ it should respect the timing constraints stated in the transition executing in $T_i$ (taking the time of communication between only components and not through testers).

The complete testing methodology is illustrated in Fig. 6.
6. Conclusion and future work

Robustness testing for real-Time component-based systems is discussed in this paper. To the best of our knowledge, quite a few works has been done in this field. In this paper, we present a methodology for testing robustness of real-time component-based systems using automatic fault injection and adequate distributed test architecture. Each component is described by its nominal specification and its degraded one. Three main contributions are noticeable in this method:

The first is the automatic generation of test sequence set for each component from its nominal specification. Then relevant faults are inserted randomly in these sequences in order to simulate hostile environment.

The second contribution is that by the end of the first phase of the test execution, we are able to tell about all robust-implemented stand-alone components. In this phase, we experiment mutated test sequences on each component of the SUT and we record the results traces. These later are checked on the degraded specification. Each component is said to be robust if the verdict of experimentation of record on the degraded specification gives the verdict success.

A third contribution is the test-execution algorithm that executes and synchronizes test sequence execution on local testers. The synchronization is done via two atomic statements, Signal() and Wait(), and a set of input-output queues. A queue is attached to each tester. The robustness of the whole SUT is deduced if the communication events between components are accepted by the degraded specifications of all components.

As a future work, we intend to investigate more realistic hazard insertion by using metrics produced by real case studies. We need also to experiment this methodology in a real case study such as industrial control software or complex embedded systems.

References


Complete management scheme for intelligent terminals for the information super-highways and its design

Hazem El-Gendy\textsuperscript{a,}\textsuperscript{*}, Nahla El Zant El Kadhi\textsuperscript{b}, Osman Ibrahim\textsuperscript{a} and Narayan Debnath\textsuperscript{c}

\textsuperscript{a}Faculty of CS & IT, Ahram-Canadian University, Egypt, 6th of October City, Egypt
\textsuperscript{b}Ahlia University, Bahrain
\textsuperscript{c}Computer Science Department, Winona State University, Winona, MN, USA

Abstract. An Intelligent Network (IN) is a general concept network command architecture to enable the rapid introduction of new services by rearranging the basic functions of transport information mainly (voice, data, and images). Modern telecommunication networks are complex distributed systems that require high monitoring level using specific management network tools.

An IN management scheme for ISO Interface is presented in this paper. The interface ensures different interactions through a set of services called Input/Output Services between desktop applications and voice terminals. Such services are designed in order to allow computer applications to provide users with enhanced features tacking advantages from the wide variety of telephony device characteristics and types.

Basically, we suggest a software switch-based interaction scheme. The switch-based software object called Information-Path, is associated with the device that allows input to be received from the device and output to be sent to the device. The management scheme is suitable for multimedia interactive telecommunications. It covers applications that involve only voice calls (Integrated Services Digital Network (ISDN) B channel), only data calls (ISDN D channel), or both voice calls and data calls (both ISDN B channel and ISDN D channel). This makes the proposed system suitable for multimedia communications that involve voice, data and graphics. The feasibility and applicability of our proposal is demonstrated through a set of applications scenarios that involve multimedia mails exchanges.

Keywords: Intelligent networks, multimedia applications, desktop applications, computer and communications protocols

1. Introduction

The current network connectivity has increased over the last decade. An explosive growth in the number of computers and the need to share information has triggered the development of computer networks. The development of information super-highways is fast spreading all over the glob. All types of information (voice, fax, data, video, and image) are being daily exchanged over a wide range of networks and super-data highways. The last decade, has been marked by the introduction of a set of multi function devices leading to the convergence of Voice Terminal (telephone), Data Terminal (computer), and Television [4]. Also, Voice Terminals, particularly multimedia Terminals, are becoming ever more advanced and equipped with multiple keypads and built-in video displays. These video displays range...
from a simple display that supports only characters on one line (e.g., to display the caller’s telephone number) to a display that supports multiple lines with several characters each (e.g., executive voice terminals). Eventually, the capabilities of these terminals’ displays will evolve to fully support the capabilities of a data terminal. Multimedia Terminals typically support two communications channels; a D Channel and a B-Channel.

The D-Channel is usually used for signaling of control information and for low rate data. The B-Channel is used for voice communications and mainly high data rate communications [4]. The emergence of information super-highways will strengthen the need for more efficient reliable and flexible communication services provided by the IN concepts protocols and products including the creation of separate control facilities [9–11]. Furthermore, multiple applications may have access to the user’s (multimedia) terminal simultaneously. For example, the terminals screen may display the caller’s telephone number; any information pertained to the current active call, general information or the user’s electronic email. Such variety and facilities requires the developing of specific scheme for efficient access management to such vital resource. One of the crucial issues here is related to application priority definition. For example, what application or class of applications should have priority in accessing the user’s terminal? How to transfer/switch access of the terminal from one application to another? How can this be achieved while ensuring a high level of flexibility and user control of such properties?

In this paper, we present a management scheme for a multimedia-IN Interface between a Telecommunication Switch (e.g., a Private Branch Exchange or a Central Office) and a Computer. The suggested scheme ensures interactions, via a set of services called Input/Output Services. Services here are defined/ensured between desktop applications and voice/multimedia terminals. Such services allow computer applications to provide users with enhanced services by tacking advantages from the wide variety and characteristics of a large and continuously upgraded telephony device types (e.g., devices with built-in displays).

The rest of this paper is organized as follows. In Section 2, we present Intelligent Network Definition. Section 3 presents the Intelligent Network Objectives. The Intelligent Network Components are presented in Section 4. Then we present the Interface Capabilities in Section 5. We then develop the management scheme in Section 6. In Section 7, we develop Application Scenarios for the Interface. We conclude the paper in Section 8.

2. Intelligent network definition

The IN is a concept that has nothing to do with artificial intelligence and neural networks. Intelligent Network expression actually appeared in the area of network architecture and in particular to the switched telephone networks [16]. This term covers a number of architecture change proposals to allow easy introduction of new services and therefore making the network more sophisticated, more “intelligent”.

The IN term is used to describe a concept of architecture which is intended to be applied to all telecommunications networks. The term "services" must be interpreted in a restrictive sense: they are built on basic information such as voice, data, or video transport services (telecommunications) network and services. Therefore, it does not cover physical resources to transport information services.

3. Intelligent network objectives

The main objectives of IN can be grouped as following:
– Facilitate the introduction and editing of new services with significant associated development time reduction.
– Reduce development costs.
– Introduce more sophisticated functions in the network, such as allowing users to manage and edit their own data.

4. Intelligent network components

Intelligent Networks are composed of switches, databases and specialized servers, in addition to the transport networks. Intelligent Network architecture includes two aspects:

– Centralized part of intelligence: data relating to a service and the processing logic are centralized outside of the switch in specialized nodes called Service Command Points (SCP). Service Order Points (SOP) consists of two parts: the SCP for the treatment of calls and SMS (Service Management System) for placing formatting information that will complement the network database. It provides a link with the commercial network operator and/or the service subscriber. An SMS for an advanced intelligent network includes a plurality of gateways in communications with a plurality of network nodes and service clients which may issue service requests.

– Standardized interfaces: SCP service point is accessible through switches services. Modularity (temporal electronic exchanges) enables the introduction of this access services by a simple addition/plug in of software. The accuracy level of available standards leaves a significant margin for interpretation by manufacturers and operators, against a single solution.

Intelligent networks concepts theoretically can be applied to any network. These concepts are applied till now, to fixed telephony networks, and sometimes data transmission, and more recently to mobile networks (GSM) telecommunications and analog networks. The concept of Intelligent Network is designed to facilitate the creation of new services, providing operator flexibility, easy user access to available services and dynamic and useful service updated and modification.

5. Interface capabilities

The interface allows desktop applications to access the voice terminal of an end network telephony user. Such access involves sending information (data characters or video images) to the voice terminal as well as collecting data from the voice terminal (user inputs the data via the keypads of the user’s voice terminal).

This information exchange is commonly achieved over a logical communication path called InformationPath. The application can also use/update this InformationPath by invoking one of the following service functionalities:

– StartInformationPath,
– StopInformationPath,
– SendInformationPath,
– SuspendInformationPath,
– ResumeInformationPath, and
– FastInformation.
We give a brief definition/overview of each one of those functionalities. First of all, lets clarify the meaning of an InformationPath: An InformationPath is a switch-based software object associated with some physical channel to a specific device (voice terminal) which allows input to be received from the device and output to be sent to the device. An established Data-Path may be in an active state where information can flow in either direction or in an inactive state where the flow of information is suspended.

1. **StartInformationPath** (Computer → Switch, and Switch → Computer): The Start Data-Path service starts an InformationPath on the specified device. It is by default a bi-directional service. Starting a device’s InformationPath means that the computer has requested getting control of the device either on its own initiative or responding to the request of the device. When a switch-based Input/Output application invoke the InformationPath (in association with a device), the action may start by informing the computer about the invocation request. In fact, this informative step depends upon the switch application administration properties that allow or not ‘automatic starting’. If the switch can yield the InformationPath, the StartInformationPath operation will lead to starting and activating the InformationPath. If another application uses the voice terminal, the access may be denied and the request ignored. The server verifies that the service request is correct syntactically and semantically then notifies the client by sending either an Acknowledgment or a Rejection notice.

2. **StopInformationPath** (Comp. → Switch & Switch → Comp,) The StopInformationPath service terminates the InformationPath. This could occur even when the InformationPath is not active. The switch may stop the InformationPath if the user requests access or use of another Input/Output application. The switch may also stop and destroy InformationPaths in exceptional cases, such error recovery situations. In those specific cases, a particular notification message may be generated (StopInformationPath). The StopInformationPath service terminates the InformationPath’s session also when another user application is requesting a new an InformationPath to the device with a higher priority.

3. **SendInformation** (Computer → Switch, and Switch → Computer): This service is used either to write, on video displays or audio channels, or to inform the computer about a user data input (at the user’s voice terminal level for example). The service ensures information transfers between the client and the server. The result sequence sent in the response to this message indicates that:

   (a) The message was received correctly,
   (b) None of the enumerated error conditions occurs,
   (c) Processing of the information is done assuming that the received information is correct. That is why the desktop application must know the precise characteristics and properties accepted by the specifically used of the switch SendInformation feature. SendInformation may in fact be used to send information to a video display or an audio channel. The server verifies that the service request is correct and notifies the client application in order to either acknowledge or reject the service request.

4. **SuspendInformationPath** (Computer → Switch, and Switch → Computer): The SuspendInformationPath service informs the computer that the InformationPath was suspended by the switch. The computer may use this service to suggest that the switch suspends the InformationPath. The server may use this service to inform the client function that the InformationPath is suspended. When used by the client, the server, and upon receiving this service will suspend the InformationPath, The server verifies that the service request is correct and notifies the client application in order to acknowledge or reject the service request.
5. **ResumeInformationPath** (Computer → Switch, and Switch → Computer): The *ResumeInformationPath* service informs the computer that the InformationPath was resumed by the switch. The computer uses this service to ask the switch to resume the InformationPath. The server uses this service to inform the client that InformationPath has been resumed. The server upon receiving this service resumes the InformationPath. The server verifies that the service request is correct and notifies the client application in order to either acknowledge or reject the service request.

6. **FastInformation** (Computer → Switch, and Switch → Computer): The *FastInformation* service starts an InformationPath for a limited time. In fact it allows activating/starting and InformationPath for only one session. A session is defined here as the required time for sending only one message/information. Basically this service behavior is equivalent to the succession of calling the following services: *StartInformationPath*, *SendInformation*, and finally *StopInformationPath*. A desktop application must know the characteristics of the object/device using the *FastInformation* service in order to be able to use it correctly in any circumstance (e.g., displaying information on the device’s screen). The *FastInformation* service is by default bi-directional. *FastInformation* service calls may be initiated by the switch or by the computer side.

### 6. Management scheme

In this section, we develop a management scheme for scheduling the access privilege to the capabilities of the multimedia terminal (display, speaker, B Channel, D Channel). The management scheme has to:

- ensure meeting and answering the requirements of various types of applications,
- avoid imposing specific priority or restrictions to the user,
- ensuring to the user a high flexibility level when dealing with dynamically switching from one application to another.

The management scheme has to cover two significantly different channels a D channel and a B channel. Furthermore, the B channel is typically used for voice and high rate data communications while the D channel is typically used for low rate data communications. Consequently, the *InformationPath* components of the management scheme will vary depending on the type of the user’s terminal. Also, for each management scheme, there may be multiple management scenarios depending on the offered application functionalities, features and operations. Let us, consequently discuss the different and useful components of the *InformationPath*.

#### 6.1. Components of the InformationPath

An *InformationPath* provides the application with the possibility of communicating with a specific user component. Thus, *InformationPath* varies depending on the device physical capabilities and types (user’s terminal). In fact we have:

1. **Display Sets**: This category of sets has video display hardware built into the set. Such video displays are controlled through a signaling channel (D channel) which can function independently from the voice (B channel). An active *InformationPath* in this case is composed of the display, the softkeys, and the DTMF (Dial Tone Multiple Frequencies) pad. Any other key such as the line appearance keys, volume keys, speaker and microphone keys, superkey and hold key are excluded from the active *InformationPath*. These keystrokes are processed within the switch’s domain.
2. **Non-Display sets:** Applications operating on these sets are commonly voice applications that typically require a method of placing a call and reading the DTMF keystrokes from the set. Consequently, an active InformationPath is composed of the DTMF pad (or a tone receiver) and a tone generator. All non-DTMF keys are excluded from active InformationPath. In fact line appearance keys, volume keys, speaker and microphone keys, superkey, and hold key are excluded from the InfoPath and are processed within the switch’s domain.

3. **Trunks:** The applications operating on trunks are similar to those operating on non-display sets. We have mainly here voice applications which typically require having calls in progress while receiving keystrokes values. Hence an InformationPath for a trunk consists of a tone generator and a tone receiver. The switch will attach and de-attach tone generators and receivers as the InformationPath is activated and deactivated. As for non-display sets, the switch may suggest the allocation and/or de-allocation of scarce physical resources, such as tone generators, while an InformationPath is active by sending a Status command to the switch.

6.2. **InformationPath management policies**

In this section, we develop the management policy for various types of terminals for various application classes.

6.2.1. **Display sets**

6.2.1.1. **InformationPath with associated call (D channel plus B channel)**

If an existing InformationPath has an associated call then, InformationPathes are activated or deactivated in relation to such a call. Thus, the application InformationPath logically replaces the call processing InformationPath (display) for its associated call. The application InformationPath is activated whenever the call processing display would normally have appeared for the call. When the terminal is displaying call processing information for another call then, the application InformationPath is deactivated. Application InformationPath will be deactivated, for example, when the associated call is placed on hold, or when a conference call is being established, or when the superkey menu is active. The application InformationPath will be active when the associated call is in connected state and the superkey menu is not active.

6.2.1.2. **InformationPath without associated calls (D channel only)**

These InformationPathes are activated according to the same rules by which the superkey applications are activated. Since the superkey menu may be activated during both idle and non-idle states, these D channel applications may also be activated (from within this menu) during most call states. These InformationPathes become active or deactivate according to superkey application behavior. The switch may issue a StopInformationPath for an InformationPath when the switch’s superkey timer expires for the corresponding inactive InformationPath.

6.2.2. **Non-display sets**

6.2.2.1. **InformationPathes with associated calls (B channel full call)**

If an existing InformationPath has an associated call then, the InformationPath is activated and deactivated according to such call. For example, the InformationPath is deactivated when the corresponding call is placed on-hold and is activated when the call is in connected state (retrieved from hold).
6.2.2. InformationPathes without associated calls (B channel half call)

These InformationPathes are handled according to the same rules by which feature programming occurs on non-display sets. Feature programming cannot be placed on hold and so does B channel half call applications cannot be deactivated. Thus, when such an application is in progress and the user answers a call, the switch will stop the InformationPath rather than suspend it.

6.2.3. Trunks

6.2.3.1. InformationPathes with associated calls (B channel full call)

If an InformationPath has an associated call, the InformationPath is activated and deactivated according to the traffic on the trunk. The switch will activate and deactivate the InformationPath accordingly as calls are established and cleared. The computer may suggest the deactivation or activation of an InformationPath in order to reduce the application’s demand on the scarce resources on the switch.

6.2.3.2. InformationPathes without associated calls (B channel half call)

Due to Public Network Requirements, this is not allowed.

7. Application scenarios

In this Section, we develop scenarios for the use of the different service primitives to support representative applications. These scenarios cover support of electronic mail, and automated security assistance applications.

7.1. Display set scenarios

Applications for display sets include applications that utilize a D channel only as well as applications that utilize both a D channel and a B channel. Examples for these applications are electronic mail applications (only a D channel) and multimedia mail (both a D channel and a B channel).

7.1.1. Electronic mail application (D channel only)

Electronic mail display uses only video display of the voice terminal and its keypads where the user accesses her/his electronic mail via keypads of the voice terminal and the electronic messages are displayed on the video display of the voice terminal. During an active session of this application, no voice call is active and typically, the B Channel is not in use.

The scenario can be as shown in Fig. 1. A display set user invokes a read e-mail application. The switch sends a StartInformationPath as a part of invoking the e-mail application. The InformationPath manager decides to activate the InformationPath as soon as it is started. The user is now reading e-mail text. Text is communicated by a SendInformation service (from host to switch). Input is received through SendInformation service (from switch to host). The user reads two messages and deletes the second message. Then, a call arrives. The user decides to answer the call and answers it. The InformationPath manager sends a SuspendInformationPath service request to the e-mail application. Call processing takes over the display. When the user decides to resume reading the e-mail, the switch sends a ResumeInformationPath. The user reads one more messages and then deletes it and then deactivates the application.
Fig. 1. e-Mail application scenario (D channel). Figure 1 uses the following annotations: **StartIP**: StartInformation-Path; **SI**: SendInformation; **StopIP**: StopInformation-Path; **RI**: ResumeInformationPath; **SusI**: SuspendInformationPath.

Fig. 2. Application scenario for soft key-driven voice mail. Figure 2 uses the following annotations: **CR**: CallReceived; **AC**: AnswerCall; **CE**: CallEstablishment; **SIP**: StartInformationPath; **SI**: SendInformation; **SusIP**: SuspendInformationPath; **RIP**: RetrieveInformationPath.

7.1.2. Softkey driven voice mail (B and D channels)

Annotated voice mail uses both the signaling channel (D channel) and a voice channel (B channel) simultaneously. The signaling channel is used to display information related to the way of accessing the voice mail (e.g., the function of every keypad). The voice channel is used to deliver the recorded voice messages.

**Leaving a message:** A voice mail application is activated after being call forwarded to a voice mail port or by a direct call from a speed call key. The application must have a monitor [4,6] running on the voice mail port. This monitor generates a *CallReceived* message for every call at the voice mail port and that includes the calling number information [4]. Voice mail port answers the incoming call or the application instructs the voice mail port to answer the call by using *AnswerCall* [4]. Switch sends *StartInformationPath* to the host application as a result of the call entering connected state. The switch has been instructed to start and activate an *InformationPath* on the far end’s terminal as a part of this application. The user is now talking to a voice mail while manipulating it via softkeys. Text is displayed by *SendInformation* commands from host application to the switch. Input is received via *SendInformation* from switch to host. When a call arrives, the user may continue with the voice mail. The user decides to answer the call, places voice call on hold and selects the calling line. The *InformationPath* manager deactivates the *InformationPath* Call processing now takes over the set. When the voice mail call is retrieved from hold, *InformationPath* is activated and the application re-transmits the information. The following figure illustrates this.
7.2. Non-display set scenarios: Security assistance application

In this scenario, we illustrate a security guard use and actions for building/reporting a safe progress to a central security application. The report includes the exact location of the guard and implicitly informs that there is no security weakness or problem (coming from the fact that the guard is correctly placed and activated). The guard ensure such services by dialing a particular access code using the telephone at the guard’s location. This allows the application to keep track record of the locations and time of the various guards in an economic way. This application uses only a D-Channel and follow or relay on the possible scenarios are indicated by Figs 3 and 4. Notice that the information included in the message is nil (the guard’s location is identified by the extension number of the telephone used which is already included in the message).

7.3. Incoming trunk scenarios

In this case we consider an application that wishes to receive DTMF input from trunk callers. The host starts an InformationPath on the trunk. When a call enters connected state, the switch InformationPath manager will activate the InformationPath. The switch is now directing keystrokes to the InformationPath. The host may be simultaneously presenting audio output to the user such as output from recorded announcements devices, text to speech converters or human operators. When the application has received all the input which it requires, it deactivates the InformationPath.

8. Conclusion

With the recent and accelerated super data highway emergence, it becomes interesting to take advantages from the IN protocols. In this paper, we have presented an IN protocol interface that facilitates the development of an integrated message desk and multimedia interactive applications, such as multimedia mail, for the information super-highway.

A specific emphasize and care has been made to keep this protocol interface fully compatible with the OSI model by both ISO and TSS/ITU (formerly CCITT). The proposed presents scenarios and
descriptions support, for the time being only one active InformationPath per terminal at any given point of time. However, as the capabilities of the terminal’s screen evolve to the full capabilities of a current data terminal, multiple active InformationPaths can be supported in a similar fashion to the current support of multiple windows of some Operating Systems such as OS2 and Unix. As multiple applications may have access to the device, a scheduling scheme is required. Unfortunately, most of the available common interfaces do not provide any specific scheduling scheme. This makes the scheduling scheme switch-dependent. Consequently, further research is still required to develop scheduling schemes for arbitration between the different applications that need to access/utilize the user’s voice terminal.

One of the most interesting exploration areas here is to make the scheduling scheme, as much as possible, depending or based on the application required data channel types. Thus scheduling scheme properties are highly linked or depending on the fact of having a D channel, B channel or a mixed B/D channels applications. Such dependency leads to a wide range of used resources and emphasize the usefulness of having a set of facilities and services offering a relatively easy management scheme and process. Even we proved within this paper the usability of the suggested protocol through the specification of a set of scenarios tacking in consideration the previous cited classification, we are aiming to implement such management schemes on specific devices and evaluate the scheme efficiency and correctness.

References


Smart container loading

Omar Cheyaheb Makarem* and Ramzi A. Haraty

Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

Abstract. The need to move various items from one location to another has become a primary need of most companies. Items are usually boxed and arranged inside containers to be transported from one location to the other. Most companies pay shipments by the number of containers being transported; therefore, it is in the interest of the company to minimize that number. Furthermore, some items might have properties, which force special constraints on how and where they are stored inside the container. In this work, we propose an agent based solution for maximizing the use of a single container; thus, reducing the transportation cost for the companies having their items transported, as well as increasing transportation performance for transportation companies. Separate agents are used for packing, tracking, loading and unloading items. Also, artificial intelligence is provided through the use of the PROLOG programming language to specify rules and constraints on items loaded as well as on the container itself.

Keywords: Artificial intelligence, container loading

1. Introduction

Movement of items of any type, shape, size and weight has become increasingly common among most companies; industrial and commercial, as well as among regular individuals for both business and personal reasons [2]. Items being transported can range from small items of irregular shapes having negligible weight, to large heavy items with specific shapes. Companies transport items for business reason. It is common among construction companies to transport equipment used in one site, to be able to use them on another site, instead of buying new material for each site. It is even more common among production companies to ship their products from their factories to various locations where their consumers reside. Regular individuals, as well as companies, may need to transport their possessions from location to another, most notably when moving from a residence location to another. Their possessions will include furniture, electronics and many other personal belongings. Depending on the type of the item being transported, specific properties might be associated with the items. Some items can be marked as fragile, these items will need special care when being placed, and cannot be placed anywhere when being transported. Some items might be of more value than others, and will need to be handed priority when being transported. Each item however belongs to a specific customer. These items must reach the destination specified by the customer. When having his items transported, the customer will expect his/her items to arrive at a specified date, to the specified destination. Therefore, items of the same customer are usually transported together.

Shipping companies have to deal with the many types, shapes, sizes and weights of these items. These companies are responsible for shipping the items from one location to another. They might ship items by land through vehicles, such as pickup trucks, or by sea through ships. In either case items that are
to be transported will be placed inside boxes. All items are boxed in order to provide a single shape to
deal with, ease of handling and protection when transporting [2]. Providing a single shape simplifies
the problem by reducing the infinitely many shapes that items can take into a single shape: a box.
Furthermore, some items might be too small to consider them by themselves. That is why we group them
with other items inside the same box. Boxes are also easier to move around and protect their content from
damage [2]. Some companies may have only limited amount of box types to be used to further simplify
the problem. Even when boxed, these items cannot be transported alone. The shipping companies will
usually group boxes in containers for each separate shipment. These containers will serve as the smallest
units of transportation. Shipment companies will ship a number of containers in every shipment. It
is in the benefit of both the transporting company and the customer to keep the number of containers
needed to transport this customer’s items to a minimum [2]. Since the customer pays the transportation
company according to the number of shipments taken to transport his/her items, by keeping this number
to a minimum, he/she will reduce his transportation cost. The shipping company will also benefit from
this by increasing their transportation performance. This will allow them to serve more customers in
the same amount of time, increasing their profits. In order to keep the number of containers needed at
a minimum, we are faced with the problem of maximizing the utilization of a single container. In most
cases, maximizing utilization refers to maximizing the space of the container occupied by boxes, but,
depending on the situation, it might be more interesting to maximize weight, number or other aspects of
the boxes inside this container. This problem is commonly known as the Container Loading Problem [18,
20]. This type of problem has numerous applications, mostly in the packing industry as we have seen,
but also in the cutting industry, for optimizing the cutting of wood or metal into smaller pieces [1].

1.1. Container loading and related problems

Container Loading is a classic problem mainly due to its simplicity in defining it and the complexity of
finding a solution for it. This problem cannot be solved in time relative to the number of available boxes;
and therefore, it is a non-polynomial (NP) hard problem [21]. There are a lot of variations of this problem,
most commonly the “Bin Packing” [5,8,15,17,18], “Cylinder Packing” [2] and “Pallet Loading” [12,19]
problems. By simply modifying the number of dimensions used, shapes of items placed or number of
containers to place items in, we end up with a new variation of the problem. All the variations of this
problem can be summed into a single problem called the “Packing Problem”. The Packing Problem
is simply an application of the more general “Multidimensional 0–1 Knapsack Problem” [11]. The
Knapsack Problem, which itself has a number of variations, is, in its simplest form, a maximization
problem which consists of maximizing the value of the items that can fit in the knapsack [23]. Being
multidimensional means that the problem is subject to more than one constraint, since it is a 0–1 problem,
items available can be either included in the knapsack or left behind, one cannot however take part of
that item [23]. The problem can be stated more formally as:

Given a set of \( n \) objects where each has a corresponding profit, and a knapsack, pack the list of objects
that can fit in the knapsack such that the profit sum of included objects is maximized [18,23]. The
problem is also defined mathematically in [11] as follows:

\[
\text{Maximize } \sum_{j=1}^{n} c_j \cdot x_j \\
\text{Subject to } \sum_{j=1}^{n} a_{i,j} \cdot x_j \leq b_i, \text{ for } i = 1 \ldots m \\
x_j \in \{0,1\}, 1 \leq j \leq n
\]
where \( n \) is the number of items available to be packed, \( m \) is the number of constraints that the solution must adhere to, the value of a specific item \( j \) is represented by \( c_j \), \( x_j \) indicates if item \( j \) is included in the result or not, if the item is included the value is 1 while if it was excluded, the value is 0, \( b_i \) is the maximum value for constraint \( i \) that the solution can have. Finally, \( a_{i,j} \) is an entry in a constraint matrix which contains the value of each item \( j \) on each constraint \( i \).

For the container loading problem, we are maximizing the value of the boxes loaded in the container [12]. This value may vary depending on the aspect we want to maximize. Sometimes, we need to maximize the capacity of the container used, so we try to maximize the volume of the boxes chosen [12]. In other cases we might want to maximize the value of the weight inside the container, so we focus on the weight of the boxes inside the container [3]. In some other cases, we might even need to maximize the number of the boxes to be packed. So this value will depend on the logic we intend to follow. The container is subject to many constraints [3]. Most notably, the length of the container on each of the three dimensions must not be exceeded by the largest sum of adjacent boxes’ length on that dimension [2]. Another recurring constraint is that of the container’s weight: the sum of the weights of the items inside the container must not exceed the weight capacity of the container [3]. Many more complex constraints can be added to the container, such as load balance of boxes inside the container or constraint on location of objects inside of it. The container is not the only subject to constraints. Some boxes may be constrained by the weight they can carry inside them or on top of them. Some boxes may not even allow items to be placed on top of it at all [3]. In other cases, the direction of the faces of the box might even be constrained. This problem cannot be solved since it is a NP-Hard problem. In the next section we discuss how such problems are usually handled.

1.2. Methods solving the problem

Being a NP-Hard problem, one cannot find the optimal solution for this problem in polynomial time; i.e, time relevant the number of boxes we have as input [25]. In order to deal with such problems, people usually revert to Heuristics [12]. Heuristics provide near optimal solution for large number of instances [18]. A heuristic is an algorithm that does not get the optimal solution for the problem; it gets a solution close to the optimal one in good and reasonable running times. In many problems, including the container loading problem, it is more practical to use heuristics to find solutions almost as good as the optimal one in an acceptable amount of time, then to wait indefinitely till we find the optimal solution for a problem [18].

The remainder of the paper is organized as follows: Section 2 provides a review of literature related to container loading and the use of heuristics is made, showing how others used heuristics and other techniques to solve this problem. Section 3 contains the proposed solution for this problem. Section 4 discusses the results of this solution and chapter 5 serves as a conclusion.

2. Literature review

As stated before, container loading problems can be solved in acceptable times only through the use of heuristics. All the work related to this topic in the literature review consists of proposing a new heuristic or choosing an existing one and extending it, applying this heuristic, and showing its performance over the problem.

The simplest heuristic to use would be the greedy approach [12]. In this approach, boxes are chosen in a greedy manner, starting with the most desired box and trying to fit it in the container, then going
through the remaining boxes following the same approach until all boxes are chosen or the container is full. In practice, since the objective is to maximize the value of boxes in the container, the most desired box to pack would be the one with the highest value. If not packed early, these boxes will be awkward to pack at later stages. This approach is very simple to implement and very fast in practice. However, it hardly yields near optimal or even acceptable solutions. In their work, A. Lim and X. Zhang [12] extended the simple greedy approach by suggesting some “trouble-making” elements: boxes that are awkward to pack. These elements will force the use of dynamic prioritization between iterations in the algorithm. The “trouble making” aspect is represented through a priority factor assigned to each box type, this factor is dynamically changed to increase or decrease the priority of a specific box. The greedy approach is still followed, but here the boxes with the highest priority, not the ones with the most value, are being chosen.

Using this approach the most “trouble making” boxes are always placed at the bottom of the container. The idea is to get rid of these difficult to place boxes at the beginning, in order to deal later with easier boxes to place. The issue in this approach is that it does not build a layer on which other layers or boxes can be placed, rendering the placement of boxes on higher levels difficult. Common approaches to solve this problem include the wall-building approach [18]. In this approach the container is filled with a number of layers across the depth of the container. The same approach could be used to fill the height of the container instead of the depth and is called the stack building approach. The depth, or height, of a layer in either approach must be well chosen. Usually, boxes are sorted based on their smallest dimension, the box with the largest smallest dimension is then chosen, and the depth is chosen equal to the largest size of that box. These boxes will be difficult to accommodate later, so are chosen first. This wall building method also follows the greedy approach and is bound to the same weaknesses.

[18] extended this greedy approach of wall building to include back tracking, by using a tree search heuristic. The tree search algorithm is used to find the layer depths that will provide the best overall filling. Even though not all possible branches are studied, due to computational complexities, the heuristic has some good results. The wall building approach, whether using the greedy or the tree search heuristic, is designed to fill containers with boxes based on their volume. This approach is not well suited to fill the container based on other aspects of the boxes, such as their weight or their priority.

Linear programming is another common method used in heuristics for the container loading problem [20]. Linear programming is used to find the best outcome; maximum or minimum, for a linear function given equality and inequality constraints [13]. Linear programming is mostly used for optimizations in the operation research, microeconomics and business management. In his work, Scheithauer [20] was able to find tighter bounds for the container loading problem through the use of linear programming relaxation. Scheithauer argued that the three dimensional packing pattern of the container can be described by smaller spatial dimensions. Through the use of a single dimension: length; the packing pattern of the container can be described through a set of patterns on this dimension, called the bar-patterns. The container is thus partitioned into bars having $1 \times 1$ cross-section (1 unit in height and length). The real position of the bar is not restricted and is said to be “relaxed”. These bars contain parts of the packed boxes and can be characterized by integer vectors and are subject to constraints. The problem becomes a linear programming problem by adding the objective function to be minimized: the function showing value of packed boxes. To achieve a tighter bound for the container loading problem, the linear programming problem must be solved. Similarly, Scheithauer was also able to describe the packing pattern using a set of two dimensional patterns. This method also works for the multiple containers. The drawback of this method is that it only considers orthogonal packing of the boxes, while in reality; the box has six orientations to be considered. Boxes with same sizes and shapes but with different orientations are considered boxes of different types.
As opposed to the previous methods; which deal with boxes themselves to find the solutions, researchers can resort to search for the best solution in the solution space. This can be achieved through the use of meta-heuristics [16]. The most widely used meta-heuristics include local search, branch and bound, and genetic algorithms.

Local search iteratively searches solutions moving from a solution to one of its neighbors while modifying the neighborhood structure as the search progresses, until some criterion has been satisfied [14]. Tabu search [22] is a special type of local search because it uses a Tabu list, containing list of solutions visited in the recent past. The solutions in the Tabu list are excluded from the neighborhood of the current solution.

Branch and bound also searches the complete solution space. It uses branching to divide search space into subspaces recursively, and bounding to find the upper and lower bounds of the studied branch [4].

Genetic algorithms alter the pool of solutions by combining or mutating existing solutions, better solutions survive while solutions of lower quality are discarded. The process is repeated until an acceptable solution is found [3] used genetic algorithms to add the load bearing aspect to the problem.

In their work, [19] used Tabu search heuristic to solve the pallet loading problem. The method proposed generates an initial solution using block heuristics, then apply block expansion moves. In block heuristics, block patterns are formed out of smaller boxes arranged in the same orientation. In block expansion moves, neighboring blocks can be decreased in size, divided into one or more blocks, or grow in size in order to preserve the solution feasibility.

[15] developed an exact algorithm with a continuous lower bound to fill a single container. Their algorithm starts by solving the problem in two dimensions, then uses a branch and bound algorithm to develop it into a container loading problem solution: All the boxes placed on the lowest level are then used as bases for sub-containers ranging up to the ceiling of the container, and the problem is repeated for this sub container. This algorithm can also be applied to the bin packing problem. Although this algorithm is easy to implement and has a lower bound, it does not perform as good as other studied algorithms.

[8] solved the bin packing problem. Their solution first solves the problem using the greedy approach then iteratively decreases the number of bins using a guided local search algorithm. This algorithm also restricts the orientation of the boxes.

3. The solution

In order to solve the container loading problem, rules and constraints to abide by need to be set. The problem is also subject to some assumptions that must be stated. Assumptions, rules and constraints set the frame for the solution. The solution is based on four agents: one for each of the packing, tracking, loading and unloading activities. While packing, items are placed inside boxes and the constraints and rules are set. Tracking locates items in boxes, boxes in the container and the location of the container itself. The loading activity involves loading boxes in the container. The logic of the loading activity can be modified by specifying the logic to follow through the use of PROLOG. Unloading deals with removing all the boxes from the container, finding the fastest way to a box in the container and the lessons learned from the loading activity.

3.1. Assumptions, rules and constraints

By definition, the problem only deals with boxes. It is assumed that all items to be transported are packed in boxes that can fit them. Smaller items can be grouped and packed in a single box. This means
that a box can contain more than a single item. These assumptions simplify the problem by eliminating
the need for smaller boxes for the small items or having a larger empty box. Once a box has been placed
in the container it is assumed that it will stay still in that location for the duration of the shipment. In
practice, this can be easily achieved through the use of ropes or by filling the empty space of the container
with polystyrene for lighter items or wood and rubber compartments for heavier items. Items placed
inside the boxes are also assumed to stay still during the shipment. The idea is preserve the items that are
being transported. This can also be easily achieved through choosing the right box sizes for the items and
holding the item in place by filling the empty parts of the box with polystyrene or rubber, if necessary.

Boxes can be made of different material and sizes. Boxes made of the same material and sizes on all
dimensions are considered of the same type. It is assumed all boxes of the same type have the same
characteristics such as the weight that they can carry in or on top of them. When boxes are packed, it is
assumed that a box cannot hang in the air. In practice, it is very difficult or even impossible to place a box
without supporting boxes under it. Each box must be on the floor of the container or on top of another
box. All boxes must be packed orthogonally. This rule forces the faces of the boxes to be always parallel
with those of the container. In practice, if the box is placed in an inclined position it is highly probable
that it will slide and fall and damage its content or the content of other boxes. This last rule takes care of
such situations. Because boxes are placed orthogonally, a box only has six possible orientations. In each
orientation the different sides of the box are parallel to different sides of the container. However, some
boxes may require a specific face of the box to be facing a specified direction. This is very common in
practice; boxes are marked with “This side up” tags to specify orientation. When boxes are placed in
the container they can be placed next to each other or on top of each other but without overlapping. If
boxes overlap, the box itself, and possibly its content, will be damaged. Each box has a limit weight
that it can carry on top of it. This weight is assumed to be maximum weight the box can carry without
being damaged or deformed; otherwise, the boxes will overlap. All boxes of the same type are supposed
to have the same limit. Some boxes might be so fragile that they require that no other boxes be placed
on top of them. Box types also have a limit on the weight they can carry inside the boxes. In practice,
if the item inside the box is too heavy, even if it fits in the box, it will damage the bottom of the box.
Therefore, items must be place only in boxes that can carry them.

In order to find where items and boxes are in the container, every box and item is tagged with a unique
serial number. The serial number is used to identify the location as well as the owner of the items. The
location of the box in the container may need to be restricted. In practice, some items might need to
be placed on the floor or next to a side of the container for support. Sometimes a box with a very high
priority might be restricted to stay close to the entrance of the container for easy access. The container
itself may have restrictions on the location of boxes. In the case where we have boxes belonging to
different customers, it might be required to put the boxes of the same customer near each other. Here, the
container will place a constraint restricting each customer’s boxes to a specific location. The container
may do load balancing on placed boxes. If the container is a truck, it is required that the center of gravity
of the container be close to its middle, otherwise the truck risks flipping over.

All the assumptions just stated are assumed when the solving the problem. All specified rules and
constraints must also be satisfied. The assumptions, rules and constraints constitute the environment
in which work is done to solve the problem. Next, in the environment just set up, the use of agents is
discussed.

3.2. Agents

As stated earlier, the container loading problem is used in practice to simplify transportation of items
from one location to another. Transporting an item in a container involves several steps and underlying
processes. First, it involves packing the item into a box so it can be placed into the container. Here, we refer back to the assumption that all items transported must be within boxes. Second, the box must be loaded into the container along with other boxes, in a way that optimizes the use of the container. Finally, the box needs to be retrieved from the container when it reaches its destination. During all the steps, location of the box and the container must be known; therefore, they should be tracked. In order to automate the transportation of the item, we deal with agents responsible for the processes involved. An agent is software that assists, or acts on behalf of, the user in performing repetitive computer related tasks [9]. Agents apply reasoning capabilities to reach a conclusion. Agents are a form of artificial intelligence where machines imitate human thinking. They are used to replace the tedious and repetitive task that a human has to do every time a set of items needs to be transported. When given the logical thinking to follow, the agent will come up with the same result a human would in much less time.

For each of the four processes involved in the transportation of the item, an agent has been created. The agents share data about the boxes, containers, items and other information through a database that they all use. The database stores information about the box types, boxes, the container itself and the items being transported. The database identifies which items are in which boxes; it also specifies the type of a box and its location inside the container. For cases where we have more than one container, we can determine which container contains the box. Furthermore, the database includes special rules and constraints called properties. Finally, the database is responsible for the association of these properties with boxes, box types and the container.

3.2.1. Packing agent

The packing agent acts as an observer of all packing operations and activities. It notes all the information by adding them to the database. All items, boxes and containers to be used in the problem must be identified by this agent at this stage. All items identified must have a name, a description and a weight. Items names should be unique and would usually include owner’s name to be found easily by the tracking agent. Description is optional and only provides additional information about the item if necessary. It is assumed that all items are weighted before being transported. If an item cannot be weighted easily, its weight would be estimated. Boxes used must be of a specific type. Each type specifies the box’s height, length, width, material, weight, weight limit inside and weight limit on top. In practice, transportation companies would have only a limited number of box types, which they either buy or manufacture. Depending on that type, the dimensions, weight and weight load of the box changes. Each box, besides being of a specific type, must have a barcode, destination, owner and description. Barcode is a unique identifier of the box for the tracking agent. This helps locate the box easily. Almost always, all items in the box belong to the same owner; therefore, it is safe to set the owner at the box level. Even though boxes contain items for the same owner, sometimes, the owner wants items shipped to different locations. Furthermore, the destination of the box must be known to know in which container it should be loaded. Description is again optional only for the boxes that need additional information. After identifying all the items and boxes to be used, the packing agent notes into which box every item is being placed. This will allow the tracking agent to identify what every box contains and in which box every item is contained.

Next, the container in which the boxes will be placed must be specified. A container has a name, description, destination, height, length, width, and weight capacity. In practice, the container could be the back of a truck or a storage room in a ship among others. The container is identified by its name. Additional information of the container can be provided through its description. The dimensions of the container can be easily measured, while its weight capacity is usually estimated. Constraints and rules
can be added to the problem through the use of properties. Properties can be applied to items, boxes, box types and containers. Properties have a name, a description and a restriction. The name identifies the property. Additional information can be provided for the property through the description. The restriction includes the rule or constraint that restricts the loading operation.

All properties to be used are identified by the packing agent. The agent also assigns these properties to the container, box, box type or item. These properties will be used by the loading agent to enforce rules and constraints on the loading operations.

3.2.2. Loading agent

The loading agent is responsible for loading the boxes in the container. This agent will take the list of available boxes in the database and try to load them into the container. The loading logic varies through two Prolog files, one for the Prolog facts and the other for Prolog queries.

Prolog is a programming language used for artificial intelligence programming [6]. Like all other artificial intelligence software, Prolog attempts to imitate human behavior by attempting to understand and analyze information and knowledge. For Prolog, there are two main types of knowledge: facts and reasoning procedures. Facts are information that is known to be true, while reasoning procedures, or queries, follow reasoning of facts [7]. The .Net framework is used to create the agents. In order to read Prolog files using this framework, a library called P#, created specifically to bridge between .Net and Prolog, is used. The loading agent works by retrieving the loading logic from the Prolog facts and queries, then working its way through the boxes available in the system and tries to fill them in the most optimal way. To do so, the P# is first used to load the facts file, and the queries are then executed against these facts. The answers to these questions are reported in an output file. The agent will read the results from the output file, and according to these results it changes its logic and behavior. According to the logic specified in the Prolog file, the agent will sort boxes by specified criteria. The agent will try to insert these boxes according to their order into the container.

The loading is achieved through the use of sub-containers. When the problem starts, there is only one sub-container, which is the container itself. Every time a box is inserted in a sub-container, it creates three sub-containers formed by the volume between a box’s faces and sub-container’s faces parallel to them. If a sub-container is too small to fit any box, it is discarded. Sometimes, when a box is placed, it intersects existing sub-containers and thus reduces their size. The program continues until the smallest box is larger than the largest sub-container and no more boxes can be placed in the container. The resulting load is the solution to the problem. Loads can be temporarily stored and continued later. This happens when a list of boxes is inserted and a load is executed on these boxes, then more boxes are added to the system. In this case, the user has the option to continue loading the remaining unloaded boxes in the container while keeping already loaded boxes in their places, or to remove the loaded boxes from the container and repeat the load with all the boxes in the system.

3.2.3. Unloading agent

The unloading agent is primarily used to unload boxes from the container, and items from the boxes. Just like the packing agent is an observer of packing operations, the unloading agent is an observer of all unloading operations. Every time a box or item is unloaded, the agent notes the change in the database. The unloading agent uses features from the tracking agent to get the list of boxes blocking a specific box. This agent determines the easiest and fastest way to get to that specific box, and the items inside it, by determining list of boxes that need to be removed first. While unloading boxes, the agent can add lessons learned in form of Prolog facts and queries. The packing agent adds constraints and rules
imposed by the user through properties. The unloading agent on the other hand changes the logic of the loading activity for future loads by adding facts and queries about the last load. Facts might include lessons related to the location where the box should be placed, such as placing wooden plates on the top of other objects and not on the floor. The queries that might be added will change the logic of the loading operation. Queries are questions that will be asked and checked against current facts, such as “should wooden plates be placed on top of other objects?”

The agent is able to clear the database. This feature is used when a new experiment is to be performed. In practice, the old data and information is usually backed up, and then the database is cleared to allow the new experiment to be performed. The agent is also able to delete all types of objects in the system. This feature is used when one box, box type, item or container were incorrectly added, or when we want to perform the experiment without the specified object. This agent is also able to remove constraints and rules applied to the problem by removing properties associated with boxes, box types, containers and items.

### 3.2.4. Tracking agent

At any point, the tracking agent is responsible for identifying the boxes, box types, items, containers and properties in the system. Although the information about these components is inserted in the packing agent, the tracking agent is responsible for retrieving this information. All the retrievals performed by the tracking agent are from the common database, making this agent the search engine of the system. Another piece of information inserted at the packing stage and retrieved at this stage is the items contained in a box. Given a specific box, the tracking agent will retrieve all the items contained in that box. On the other hand, the agent is able to find the box containing a specified item. The agent is also able to identify all the boxes, and thus, items contained in the container. Furthermore, given a specific box, box type, item or container the agent can find associated properties. Conversely, given a property, it can find all boxes, box types, items and containers that are associated to this property.

This agent also keeps track of the location of items and boxes. At any point, given a specific box or item, the agent is able to find out if it is loaded in a container, and if it is loaded, the agent can specify its exact location. Conversely, given a specific location inside the container, the tracking agent is able to find the box at that location. These features allow the unloading agent to find the list of boxes that need to be unloaded in order to reach a specific box. Other activities that agent can do include finding the number of boxes of a specific type. The agent is also able to identify unpacked items and unloaded boxes in the system.

### 3.3. Graphical user interface

In this solution for the container loading problem, a graphical representation of the loaded boxes is given. This representation is achieved through the use of Microsoft’s .Net framework and an OpenGL library for .Net.

The .Net framework is a windows component that supports building and running applications [7]. The advantage of .Net is that it simplifies development, deployment and integration with a wide variety of programming languages. In this work, C# is used as the coding language to write the agents.

C# will communicate with an OpenGL library created specifically for .Net. “OpenGL is a software interface to graphics hardware” [10]. OpenGL is responsible for sending information to the hardware that will graphically draw the solution.

The graphical user interface lists all the containers, boxes, box types, items and properties in the system. By selecting a specific object, the system will display detailed information about that object (as shown in Fig. 1).
The interface also provides controls to add new objects (as shown in Fig. 2), delete existing objects or associate other objects with properties. It also allows the user to change the Prolog facts and queries to be used in the loading process.

At any point, the user can empty the container and then refill it using the changed Prolog logic. This can be achieved through the interface by the assigned buttons for these actions. The graphical user interface will load information about the container and the boxes inside this container. The user interface will draw only the frame of the container, in order to see through it. The interface will also draw solid cubes for each box with randomly chosen distinct colors for each box in order to identify them easily. The interface allows for rotation around the container to view it from different angles. It also allows for zooming in and out of the container for a better view (as shown in Fig. 3).

When user is trying to find the shortest way to reach a box, the interface will only draw the frame of the boxes that need to be removed in order to reach that box (as shown in Fig. 4). If the box can be
directly removed, all the boxes will be drawn normally.

This interface will be used to test the loading logics that created using the Prolog facts and queries. In the next section, experiments will be performed on the user interface to test the effectiveness of the solution.

4. Result and discussion

In order to test the effectiveness and efficiency of the proposed solution, it will be benchmarked against numerical data taken from [18]. That data however does not contain material type or weights for the boxes. To solve this problem, material types were randomly assigned to each box type out of the following materials: paperboard (for carton), wood, fiberboard (for corrugated boxe) and steel. Boxes weights, maximum weight that can be carried inside and on top have been assigned to each box type depending on its material and its volume. The list of boxes used as input is detailed in the table below. The container chosen to fill these boxes in has the same dimensions of the one used in [18] data. The container has a height and a width of 230 centimeters and a depth of 590 centimeters for a total volume of 31,211,000 centimeters cubed. The 146 boxes have a total volume of 91.8% of the container volume [18] algorithms were able to fill up 90.44% of the containers volume with specific ranking rules; however some tests took up to two minutes to come up with the result. The user interface is used to enter all this data as input to the problem. Behind the scenes, the loading agent is responsible of entering boxes into the database. The container is assigned a default capacity of 1,000 kg. In order to compare the results
Table 1
Dimensions and properties of input boxes

<table>
<thead>
<tr>
<th>Box type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>85</td>
<td>36</td>
<td>68</td>
<td>71</td>
<td>28</td>
<td>109</td>
<td>94</td>
<td>32</td>
<td>52</td>
<td>42</td>
<td>97</td>
<td>106</td>
<td>105</td>
<td>107</td>
<td>39</td>
<td>29</td>
<td>31</td>
<td>39</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>60</td>
<td>69</td>
<td>26</td>
<td>100</td>
<td>31</td>
<td>29</td>
<td>52</td>
<td>106</td>
<td>115</td>
<td>68</td>
<td>41</td>
<td>59</td>
<td>53</td>
<td>68</td>
<td>43</td>
<td>56</td>
<td>47</td>
<td>41</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Depth</td>
<td>26</td>
<td>28</td>
<td>46</td>
<td>100</td>
<td>42</td>
<td>51</td>
<td>43</td>
<td>113</td>
<td>45</td>
<td>44</td>
<td>115</td>
<td>48</td>
<td>40</td>
<td>32</td>
<td>110</td>
<td>80</td>
<td>73</td>
<td>42</td>
<td>114</td>
<td>40</td>
</tr>
<tr>
<td>Material</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>2</td>
<td>0.6</td>
<td>1</td>
<td>10</td>
<td>0.5</td>
<td>0.4</td>
<td>0.5</td>
<td>1</td>
<td>0.7</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
<td>4</td>
<td>2.5</td>
<td>1.3</td>
<td>3.8</td>
<td>3.2</td>
</tr>
<tr>
<td>In Weight</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>50</td>
<td>2.5</td>
<td>2</td>
<td>2.5</td>
<td>5</td>
<td>3.5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2.5</td>
<td>2.5</td>
<td>5</td>
<td>20</td>
<td>12.5</td>
<td>6.5</td>
<td>19</td>
<td>16</td>
</tr>
<tr>
<td>Top Weight</td>
<td>20</td>
<td>6</td>
<td>10</td>
<td>99</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>40</td>
<td>25</td>
<td>13</td>
<td>38</td>
<td>32</td>
</tr>
<tr>
<td>Boxes</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>13</td>
<td>9</td>
<td>12</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Fig. 3. Container graphical view.

Fig. 4. Box inside the container.
to Pisinger’s outcomes, the boxes will be loaded empty into the container. In order to try to maximize the utilization of the container, prioritization logic is used. The prioritization logic to use is specified through the PROLOG file. The logic will order boxes based on the specified priority. The loading agent will attempt to load boxes into the container based on this order. Results of prioritization logics based on volume, weight, number of similar boxes available and length will be discussed. For these tests it is assumed that the boxes loaded cannot be rotated and must be placed as defined.

First, the weight maximization of the boxes inside the container is studied. Boxes are ordered based on their weight, with heavier boxes having higher priority. Doing so should increase total boxes weight, filling a higher percentage of the weight that the container can hold. This logic fills 92 boxes in the container which constitute 63% of the volume of the container and 20% of the weight that the container can hold. This finding is rather expected since this logic would sacrifice volume for the sake of weight. Out of the studied logics, this is the logic that maximizes weight utilization of the container. This logic is not very effective when the deviation of weights of the boxes is rather small. On the other hand, it is very useful when boxes are filled with actual items of different weights.

Next, the volume maximization of the boxes inside the container is discussed. Following the greedy approach, boxes are ordered based on their volume in descending order. Boxes with larger volumes will be inserted first into the container. Running this logic will fill 73 boxes into the container. These boxes constitute 66% of the volume of the container, while their weight is 12% of the weight that the container can hold. The low number of boxes packed is explained by their large volumes, since fewer boxes are needed to fill more space in the container. The volume filled percentage is still rather low and better results can be achieved. Inspired by the wall building approach, the following logic prioritizes boxes based on the number of boxes of the same type that are available. The idea behind this logic is that boxes of the same type constitute perfect walls, reducing wasted space between them. This test fills 125 boxes in the container which constitute 67% of the volume of the container and 16% of the weight that the container can hold. Even though this logic loads the most boxes into the container, it still does not provide a much better solution then loading based on volume. The reason is that there are usually more small boxes and fewer large boxes. In this case, the smallest boxes were being loaded, increasing amount of loaded boxes but not greatly affecting volume filled.

Finally, the logic that orders boxes based on their length is studied. Here, boxes with the longest length have higher priority. This logic also imitates the wall building approach. Here the walls are defined by the lengths of the boxes. When a box is placed inside the container, it defines a wall segment having for width the length of the box. Since boxes are prioritized by their length, all smaller boxes are candidates for that wall segment. This test fills 101 boxes in the container which constitute 71% of the volume of the container and 15% of the weight that the container can hold. This logic is also similar to that of Pissingerg’s, except it is only without allowing boxes to rotate and wall segments are determined by box’s length. This logic proved to be the most efficient among the studied logics. These findings agree with the reviewed literature which shows that wall building approaches are more effective than greedy approaches.

[18] approach can also be implemented in the logic. This can be achieved by adding rotation to boxes and prioritizing based on the largest smallest dimension. Pissingerg builds his wall segments based on the smallest dimension of a box, so prioritizing boxes based on their smallest dimension should give similar outcomes to those in his research. Regarding speed, some of Pissingerg’s tests took up to two minutes to come up with results. The loading agent came up with results relatively faster. It took the agent less than five seconds to fill the container. The studied tests are all executed with empty boxes. Adding items to the boxes will increase the constraints on the problem. The resulting outcomes will be constrained by the weight and positioning constraints enforced by the items on the box or on the container.
5. Conclusion

In this work, a new way to solve the container loading problem using agents has been studied. Four agents were used to perform the various tasks involved in transporting items: a packing agent, a tracking agent, a loading agent and an unloading agent. These agents communicate with each other and share information and data by using a common database. A set of assumptions, rules and constraints have been considered and applied to the problem in order to solve it. It is assumed that the problem only deals with boxes that stay still with their content inside the container. Boxes must be packed orthogonally in one of six orientations, without them overlapping other boxes; furthermore, boxes can not be hanging in the air. Restrictions may be applied for a box’s location, or the weight it can carry inside or on top of it. The contribution of this work is in the agents created. A packing agent acts as an observer for the process of filling the boxes and assigns properties to each box or item and to the container itself. A tracking agent is responsible for locating boxes and items at all times. An unloading agent is used to unload boxes from the container. This last agent is capable of learning lessons when boxes are unloaded. It is also able to find the shortest path to reach a box in a container by identifying required boxes to be removed to reach that box. A loading agent is responsible for placing boxes in the container in a way that optimizes our required attribute. This agent that can have its loading logic changed easily through Prolog. This capability allows control over complexity and type of loading to be used. The proposed solution is rule based in that it allows easy addition of new constraints both on the container and on the box. It also allows changing the logic of filling the container, which allows it to change the complexity of the problem depending on the user’s requirements. A graphical presentation of the problem was provided to visualize the problem and the solution. The open source graphical language, OpenGL, was integrated with Microsoft .Net via a library created specifically for .Net in order to provide the graphical presentation. The discussed agents, that solve the container loading problem, are highly useful and effective. These agents need to be applied in a real work environment to fully test their abilities and discover their limitations.

Further work can be done to make these agents be used as framework for testing loading logics and visualizing the solutions.

References


Composite service metamodel and auto composition

Anthony Hock-Koon* and Mourad Oussalah
University of Nantes, LINA Laboratory, 2 rue de la Houssinière BP 92208, 44322 Nantes, France

Abstract. Service Oriented Architecture (SOA) is a computing paradigm which promises to provide transparency to resource access by exposing the available resources as services. SOA is based on service composition for application development or service development. Numerous researches focus on service composition. Each of them tries to solve some particular problems. This paper intends to reify the relevant notions of these models in a Composite Service MetaModel. This metamodel defines all interlaced features and provides a global and explicit vision of the service composition. Moreover, this approach allows the specification of the auto composition process: the composite’s ability to dynamically modify its architecture and its composition logics according to the environmental context.

Keywords: SOA, composite service, MetaModel, auto composition, loose coupling

1. Introduction

Service Oriented Architectures (SOA) [1,2] are a software development paradigm that we can summarize by “an homogeneous exposition and use of heterogeneous resources”. SOA is commonly used in Web Service [3], Pervasive Computing [4] or business applications in general. Typically, these environments are envisioned to embody a number of devices with a very rich set of functionalities. They dynamically use their available resources in an efficient way to provide the best possible support for user tasks. These environments are able to extract these needs and build their replies according to their own capabilities. These replies have to be robust and adaptable to system context and user context in order to ensure the continuity of service. Thereby, they have to face dynamic challenges such as heterogeneity, resource restrictions, context adaptability and so forth.

By modeling the available resources as services, SOA approach provides a homogeneous mechanism of combining these services together to realize the complex user tasks. This key mechanism is known as service composition. Many models were developed in order to introduce dynamicity in the composition process. Typically, they try to match user tasks and available services by dynamic comparisons of computer readable task descriptions and service descriptions. Most of these descriptions are enhanced with semantic informations by use of ontologies [5].

Existent dynamic service composition systems define and realize the required communications between selected services. Each of them focuses on specific problems, such as communications between heterogeneous resources, context awareness and so forth. However, an explicit coordination of all the relevant approaches still missing. A better understanding of the interlaced composition features will be
the base toward the automation of this coordination. This will allow for dynamic modifications of these compositions features.

Therefore, we present a composite service metamodel. A composite service reifies a service composition as a service. Our metamodel summarizes the requirements of a composite service so as to support dynamic by-reuse construction and for-reuse development. Moreover, we define the auto composition mechanism which allows for dynamic modifications of the composite’s architecture.

The remainder of this paper continues as follows. Section 1 deals with the motivations of a composite service metamodel and its contributions in the SOA paradigm. The composite service metamodel is presented in Sections 2 and Section 3. Section 2 presents the architectural elements. Section 3 shows the dependencies between these elements. Section 4 focuses on a composite service instantiation based on our metamodel. Section 5 presents a prototype which illustrates our approach. Section 6 discusses an other theoretical interest: the loose coupling notion which is a main notion in service composition and a continuity for our explicitation works. Section 7 concludes and discusses future works.

2. Related works

Existent formal definitions of the SOA paradigm (listed in [6]) such as [1,2] define the service composition as the mechanism which provides reusability and flexibility in software development. Typically, the service composition is divided in three parts: service discovery which discovers the set of candidate services according to the user needs, service selection which selects the most suitable services according to user preferences, and service composition which establishes the communication between the selected services to realize the user tasks. Our work focuses on this last part.

In order to illustrate the existent works in the specification of a composition of services, we use a very simple pervasive scenario which summarizes the main encountered problems. A system’s reply uses two devices: a microphone and a jukebox. The user says the wanted song. The microphone captures his choice and provides it to the jukebox which plays the suitable disk. Each device is represented by a service. This two services have different implementations and they use specific communication protocols. The microphone deals with audio streams. The jukebox requires a string as the song choice. These services use Wi-Fi as communication support.

This example illustrates a first problem which is the dynamic definition of the collaboration between reused services. Numerous researches focus on this part. These researches can have a general approach such as [7,8] which focus on modeling a collaboration, or a domain specific approach with their specific problems, such as [9,10] in Web Services (distribution, security and so forth) or such as [11,12] in Pervasive computing (resource restrictions, history informations and so forth).

An other problem is the heterogeneous nature of the services which can not directly communicate (the microphone and the jukebox have different interfaces). All reused services have to be reachable and invokeable. Moreover, the data which are exchanged between these services have to be understandable. Therefore, some approaches [13,14] focus on these interoperability problems.

An other main challenge in a service composition is its flexibility. In our example, we assume that the jukebox service becomes unavailable. The pervasive system has two other services which can ensure the same capability: a MP3 player and a speaker. The SOA paradigm aims that a composition is able to identify a defective service and remplace it according to the system context or user context [15,16]. The redefinition of a composition implies its coordination with the two other aspects (collaboration and heterogeneities). However, a model which can reuse the previous approaches and coordinate all these aspects still needs to be provided.
In order to maximize this reusability, we reify a composition of services as a composite service [1,2]. This homogeneous approach ensures:

The easy storage and reuse of resolved tasks. Identified compositions will be managed by the system as available services. Previously asked task will result in an exact matching with a composite service. Moreover, saved composite services will offer some high level functionalities which can be use for building new system’s replies (composite of composites).

And the easy reuse of all promising mechanisms of the SOA paradigm (such as service replication, migration [17,18]) to increase the overall quality.

All the previously listed features of a composition have to be unified in this single composite model. However, none of existent formal definitions of SOA ([6]: OASIS, OpenGroup, W3C, IBM, etc.) address a clear specification of the composite service. Typically, they define a composite service as a flow of information and control through the individual services. This does not explicit the previous features and the means of their coordination.

We intend to define an explicit composite service metamodel which specifies this coordination. This metamodel is a contribution toward a better understanding of the service composition features and to the requirement of consensus advocated in [6,19]. This metamodel intends to reify at the architectural level all the interlaced features of a self adaptable composite service. It is defined as a managing entity above the previous approaches.

3. MetaModel: Architectural elements

Our approach is to reify at the architectural level all the relevant composite features. Then, we introduce each architectural element of our metamodel by the associated composite feature which is clearly identified and specified.

3.1. MetaModel overview

We build our composite service metamodel as an extension of the OASIS service metamodel [1]. This facilitates the understanding of our approach and its localization among all the notions of a SOA specification.

A composite service is a service. It inherits all widely accepted elements defined in the service metamodel [1,2]. Then, a composite service is composed by a Service Description and a set of Capabilities [1] (cf Fig. 1). By opposition to an atomic service, a composite service is the result of composing one or more other services: constituent services. These constituent services represent reused services which
are included in the composition and which coordinate their actions to realize all provided capabilities of the composite. We group the constituent services in functional service or non functional service where:

- Functional service (Fservice): selected services which match the task requirements and the composite requirements. They provide their capabilities without knowledge on the other constituent services. They do not take into account the involved composition. Fservices are gathered in the Functional Composite Service or FCS.

- Non functional service (NFservice): services that handle all features related to the composition logic. NFservices have to manage the other constituent services. Their capabilities depend on the composition. Then, they have a global knowledge on the services involved in this composition. NFservices are gathered in the Composite Service Manager or CSM.

Thereby, a composite service can be summarized as functional features that resolve the task, and non functional features that ensure the composition logic. We make this distinction visible in our metamodel. Our composite service is composed of two elements, the FCS and the CSM (cf Fig. 1).

The FCS represents the features which are visible for the users. Its represents the exposed composite capabilities which can be invoked by a client of this composite. The FCS reuses the capabilities of its constituent services (Fservices) to achieve its own which are directly promoted as the capabilities of the composite. The means used to realize these high level capabilities is under the responsibility of the CSM and its NFservices. In fact, the CSM defines the collaboration between Fservices and it ensures this collaboration which realizes the high level goals.

In this work, we focus on the specification of these non functional aspects of the composite. We intend to specify all the mechanisms involved in the composition. We try to provide a better understanding on how to manage the constituent services to realize the high level goals. Then, we focus on the definition of the NFservices which encapsulate the composition logic. All following NFservices are abstractions of important composition features. First, we specify these architectural elements according to the identified features. Then, we motivate the organization of our metamodel elements. Our contributions will result in the explicit definition of their interdependencies and their coordinations during the auto composition process of the composite.

### 3.2. Composite Service Manager (CSM)

The CSM gathers all NFservices which manage the composite service. It is totally transparent to the user which only requires the functional features (FCS). CSM is invisible by opposition to FCS which is visible to the user. According to the related works, we abstract four main non functional roles:

- invocation – capacity of the composite to trigger its constituent services [13,20];
- collaboration – capacity of the composite to know which Fservice has to be invoked and to coordinate these invocations [7,8];
- mediation – capacity of the composite to ensure data exchange interoperability between its constituent services [13,14];
- adaptation – capacity of the composite to auto modify its architecture and its related composition logic [1,15,16]. According to its new architecture, the composite have to correctly modify the previous non functional roles (invocation, collaboration, mediation) to ensure its consistency.

In addition to provide a better understanding of the service composition by metamodeling a composite service, the self adaptation is an other main motivation of our approach and a contribution toward the specification of the auto composition (we define auto composition as all the required dependencies
and processes to ensure on the fly modifications of a composite). A modification in the composite architecture implies possible modifications on the composition logic. First, we associate each role with a specific architectural element which composes our CSM. Then, we define the dependencies between these NFservices which are maintained in the CSM. We define the CSM has the central non functional entity which coordinates its NFservices and which handles their consistency.

3.3. Invocation

We reify this feature at the architectural level and we define the Invocation NFservice (cf Fig. 2).

3.3.1. Invocation NFservice

A NFservice responsible for invoking its associated service. It represents the technological bridges required in SOA. It is interface-based and then its implementation depends on the service description of the related service. An Invocation NFservice can be bound to different services as long as interface constraints are respected. It builds a well-formed invocation message and it uses suitable communication technology. Therefore, each constituent service is associated with an Invocation NFservice. If no particular message implementation is needed, the Identity is used.

3.4. Collaboration

3.4.1. Collaboration manager

The NFservice responsible for the collaboration schema between Fservices. It maintains the business logic. The Collaboration manager specifies workflow and dataflow between Fservices and FCS. Workflow schedules the invocations of Fservices, establishes precedence links. Dataflow expresses the data exchanges (outputs to inputs) between Fservices and data exchanges between Fservices and FCS, establishes use links.

The Collaboration manager also defines client links from FCS to Fservices. In fact, FCS is the client of its Fservices which are not necessarily incorporated physically in the composite [21]. A Fservice can have multiple clients and it focuses on its own capabilities (as an example: Google). The FCS only reuses capabilities offered by these Fservices.

Then, we model the collaboration schema maintained by the Collaboration manager by a graph with FCS and Fservices as nodes and precedence link, use link and client link as edges (cf Fig. 3). Fservices are brother nodes; they are in the same level of composition. FCS is the parent node and all its constituent Fservices are child nodes.
3.5. Mediation

The mediation focuses on the interoperability problems between two Fservices which exchange data. Since Fservices can be heterogeneous, exchanged data can require transformations to be understandable. Then, some particular services are selected to ensure these data transformations. We call these services: Mediation service or Mservice. The Mservices are based on Fservice interfaces and they can be bound to any services which respect this constraint.

The Mservices have no knowledge on the other constituent services. They only provide their data transformation capability. Thereby, a Mservice proceeding is equivalent with a Fservice proceeding. Then, we gathered in the FCS all Mservices used in the composite. However, the Mservices are equivalent with a Fservice, they are not visible for the clients of the composite. The realization of the composite capabilities is hidden from its users. The transformation capability of the Mservices should not be promoted at the FCS level.

The relationships between Mservices and Fservices define a mediation schema: what Mservices or chain of Mservices are used to transform data between these Fservices. Thereby, the mediation schema is maintained by a dedicated NFservice: the Mediation manager (cf Fig. 2).

**Mediation manager**

The Mediation manager manages use links which represent the dataflow between Fservices and associated Mservices. If no mediation is required, the Mediation manager establishes use links with the identity mediator.

As for the collaboration schema (cf Fig. 3), we model the mediation schema by a graph with: FCS, Fservice and Mservice as nodes and use link as edges.

3.6. Adaptation

3.6.1. Adaptation manager

The Adaptation manager manages the functional architecture of the composite: the constituent services of the FCS. It can add or remove services (Fservices or Mservices) which are gathered in the FCS. The Fservices and the Mservices are the flexible part of the composite architecture. Since they do not have a global comprehension of the composite, they can be easily remove. Therefore, the Adaptation manager maintains the composition schema of the FCS which expresses its constituent services.

We model the composition schema by a graph with: FCS, Fservice and Mservice as nodes and composition link as edges.

The Adaptation manager follows typical service composition steps: discovering a set of candidates, selecting the most relevant services, and composing new services in the composite service. For resolving
these steps, the Adaptation manager reuses existent service composition systems. The defective set of Fservices or Mservices is the inputs of the composition system: the defective service is used as the description of a required service.

The new set of services defined by the Adaptation manager can imply some modifications on the composition logics which are maintained by the other NFservices. We call the management of all the impacts of an adaptation: the auto composition.

The following section focuses on the specification of these dependencies. We define the CSM as the central non functional entity which is responsible for managing these dependencies and for coordinating its NFservices.

4. Dependencies between NFservices: the central role of the CSM

The CSM is composed by the Collaboration manager, the Mediation manager, the Adaptation manager and all the Invocation NFservices which are required to invoke the constituent services of the composite (cf Fig. 2).

We define two types of dependencies between these NFservices:

– dependencies between schema: a new set of Fservices can imply a new workflow and a new dataflow. Moreover, a new dataflow can imply new data transformations. A new set of Mservices can be required. This impacts on the composition schema and the mediation schema.

– invocation dependencies: the composite service has to be able to communicate with the new set of constituent services regardless their implementations.

4.1. Dependencies between schema

We focus on dependencies between composition schema, orchestration schema and mediation schema. We have followed an homogeneous approach in order to model these schema: nodes represent services, and edges represent dependencies between services. This approach facilitates the understanding and ensures the homogeneous treatments.

Thereby, these dependencies between schema represent impacts of a modification on a schema on the others. Since we represented a schema by a graph, a dependency link expresses impacts of a graph change on the other graphs. Therefore, we define a set of admitted changes for the collaboration schema, mediation schema and composition schema. These changes are logic combinations of atomic graph operations (add/remove node/edge) which ensure the consistency of the graphs.

Then, we specified three types of dependency links based on these admitted changes:

– AdaptationToCollaboration dependency link – specified impacts of the adaptation schema on the collaboration schema.

– AdaptationToMediation dependency link – specified impacts of the adaptation schema on the mediation schema.

– CollaborationToMediation dependency link – specified impacts of the collaboration schema on the mediation schema.

We do not detail these three dependencies. However, each of them is divided in sub dependencies which define involvements between admitted graph changes. All these dependencies are handled by the CSM. Then, we define a dependency schema which is maintained by the CSM.

We model the dependency schema by a graph with: Adaptation manager, Collaboration manager and Mediation manager as nodes and AdaptationToCollaboration dependency link AdaptationToMediation dependency link and CollaborationToMediation dependency link as edges (cf Fig. 4, white items).
4.2. Invocation dependency

The CSM is responsible for the invocation feature of the composite service. It gathers all the Invocation NFservices and it knows which services can be invoked through these NFservices. Then, we extend the previous graph (cf Fig. 4). We add (gray items): an invocation link in the set of edges, and Fservice, Mservice, FCS and Invocation NFservice in the set of nodes (cf Fig. 4). The invocation link is used to bind an Invocation NFservice and its related services.

The Adaptation manager can modify the sets of Fservices and Mservices used in the composite service. According to these sets, the CSM has to define new Invocation NFservices which can ensure their invocations. Thereby, we specify these notions in the dependency schema by a set of admitted changes:

- **Functional invocation changes**: adding or removing a Fservice node and handling its related invocation link and Invocation NFservice.
- **Mediation invocation changes**: adding or removing a Mservice node and handling its related invocation link and Invocation NFservice.

For the moment, we have defined each architectural elements introduced by our metamodel and the dependency rules between them. In the following section, we define the non functional processes. We define the communications between the constituent services which ensure a running composite and the auto composition process.

4.3. Non functional processes

The CSM centralizes all non functional processes. It asks for functional orchestration to the Collaboration manager which specifies the Fservice which has to be invoked. Then the CSM uses the suitable Invocation NFservice. The Collaboration manager also indicates data exchanges. According to these exchanges, the CSM asks for mediation to the Mediation manager. The Mediation manager indicates the suitable Mservices which have to be invoked by the CSM to resolve the interoperability problem.

An other process is required to ensure self adaptation and auto composition. The CSM has to know if a Fservice or a Mservice is defective. Therefore, we add an other role to the Invocation NFservices. They are responsible for reporting to the CSM any changes in their bound service (unreachable service, decreasing response time and so forth). According to these changes, CSM triggers the adaptation process for the reported services by invoking the Adaptation manager.
Then, the CSM handles dependencies between its services during the adaptation process. The Adaptation manager indicates added Fservices or Mservices to CSM. The CSM invokes the Collaboration manager, the Mediation manager and the Adaptation manager. It triggers the suitable graph changes according to the defined dependencies: AdaptationToCollaboration, AdaptationToMediation, CollaborationToMediation and Invocation dependency.

These non functional processes are based on orchestration mechanism. In fact, choreography represents peer styles of interactions. They typically occur between trading partners that span organizational boundaries and imply strongly coupled relations. Thereby, loose coupling is ensured by orchestration mechanism, mediation between inputs and outputs, and interface-based Invocation NFservices.

5. Composite service instantiation

In this section, we use the jukebox example. We present a snapshot of the running composite service instance. This composite is used to illustrate the auto composition process.

5.1. Instantiated example

A CSM groups the NFservices, a FCS groups the Fservices. The CSM owns Invocation NFservices, one for each service that it has to invoke, a Collaboration manager, an Adaptation manager and a Mediation manager. The Collaboration manager handles the collaboration schema:

- Workflow: FCS uses first the microphone Fservice to listen the user choice and then the jukebox Fservice.
- Dataflow: the microphone transfers the user choice to the jukebox.

A Mservice, AudioToText, is required. This Mservice transforms an audio stream into a string. It ensures interoperability between the microphone and the jukebox. This Mservices is included in the FCS.

Figure 5 (Microphone-Jukebox) illustrates all required schema. Circles represent services nodes, edges represent dependency links.

5.2. Adaptation scenario

Assumptions: the Adaptation manager reuses an existent service discovery technique which performs extended mode of discovery by proposing one-to-N correspondence. According to the user moves, the jukebox Fservice became unavailable.
Following the workflow, the CSM tries to invoke the jukebox through its Invocation NFservice. According to predefined rules, the Invocation NFservice mentions the jukebox as unavailable. The CSM triggers the adaptation process and invokes the Adaptation manager. Based on its composition system, the Adaptation manager removes the jukebox from the composition and adds the new selected Fservices: the MP3 and the speaker. The composition schema is updated by functional composition change. The CSM has now to handle possible side effects and deals with dependency links and invocation links.

Having one-to-N correspondence implies workflow redefinitions to establish the new well-ordered scheduling. Moreover, this new set of Fservices induces a different dataflow based on the interfaces of the selected Fservices.

Thereby, the CSM invokes the Collaboration manager which updates client links, precedence links and use links: handling AdaptationToSequence dependency link. The CSM invokes the Mediation manager: handling AdaptationToMediation dependency link.

**Workflow:** FCS uses first the microphone, then the MP3, and finally the speaker.

**Dataflow:** the microphone provides the song’s name to the MP3 which streams the audio to the speaker. The Collaboration manager reports the specification of new use links which possibly require mediation. The Mediation manager tries to define the mediation schema following the use links and according to available Mservices. The MP3 Fservice interface is the same as the jukebox one. The previous Mservice, AudioToText, can be reused. Since the MP3 provides audio stream and the speaker requires audio stream, then the Identity is used between them.

After the selection of the new sets of Fservices and Mservices, the CSM has to correctly invoke them. Since the speaker interface is totally different from those used by the composite service (it requires audio stream and uses bluetooth standard), the CSM needs an other specific Invocation NFservice to communicate with the speaker.

Figure 5 (Microphone-MP3-Speaker) illustrates the modified schema after the adaptation process. Gray items represent modified elements.

### 6. Implementation

We are currently developing a prototype to realize our auto composition process. This prototype is implemented in SCA technologies [20]. SCA proposes a simple approach to realize SOA applications. It is associated with an Eclipse plugin (SCA Tools [23]) developed in the SOA Tools Platform Project.

SCA Tools provides a development platform which allows composition between heterogeneous software entities. A software entity is named as a SCA component which can be implemented in different technologies (Java, C++, BPEL, and so forth). These entities can be composed in a SCA composite. The SCA platform ensures the compatibilities between the implementation technologies. This ensures the communications.

We have two approaches to develop our prototype: we reuse existent SCA Tools features; or we extend the SCA concepts and we enhance the SCA platform.

#### 6.1. First approach

The first approach is based on model transformations [24]. We use our composite service metamodel, expressed in ecore as source metamodel and the SCA metamodel as target metamodel. We use ATL [25] to define the transformation rules and to execute them. Briefly, our composite service is associated with a SCA composite and its architectural elements are associated with SCA component or SCA composite.
We hard code all the features associated with the CSM and its constituent services in order to realize our non functional processes. Thereby, we implement the CSM by a SCA composite. This CSM is composed by four SCA components which implement:

- the Collaboration manager: it handles the collaboration schema and its admitted graph changes,
- the Adaptation manager: it handles the composition schema and its admitted graph changes,
- the Mediation manager: it handles the mediation schema and its admitted graph changes,
- a CSM Component: it handles all the CSM’s features (non functional processes and the dependencies between schema graphs). It uses the capabilities of the three other SCA components (its SCA references are linked to the SCA services of the three others SCA component).

The Invocation NFservices can be reified in the SCA composition nevertheless, they do not have a real implementation. In fact, their invocation ability is transparent: the SCA platform directly provides the management of different implementation technologies.

Thereby, an architect focuses on the definition of the Fservices which are required in his composite. Also, he provides the dependencies between these Fservices: he defines the collaboration schema. A discovery engine identified the existent services which match these user needs. Then, our composite service is instantiated according to these identified services. At the time of writing, the discovery engine is hand performed: we directly provide the SCA components which implement the required Fservices. Moreover, when an invocation of the Adaptation manager occurs, we play the role of its reused discovery engine. We provide the new SCA component which realizes the capabilities of the defective one. However, all the management of the graph dependencies are automated.

This first approach is semi automatic nevertheless, it allows a faster development. The resulted prototype only intends to illustrate our auto composition process and its workability.

6.2. Second approach

The second approach stills in its early development and it intends to be fully automate. We intend to provide to architects a development platform where they can instantiate our composite service in their applications. Then, we are extending the SCA Tools platform which will propose our composite service as a building block (like it proposes the SCA composite or the SCA component).

We extend the SCA metamodel with our composite service. Our composite service is a specialized SCA composite with additional components. These compulsory components represent our constituent NFservices defined by our metamodel.

Thereby, the SCA Tools will propose to developers a choice between implementing a classic SCA composite or implementing our self adaptable composite which provides additional features:

- this self adaptable composite has a better expressiveness than a SCA composite and it makes explicit the principles of mediation and collaboration;
- and it includes the auto composition process which allows the replacement of defectives services.

The auto composition process relies on a discovery engine which is used by the Adaptation manager. A developer which implements our composite will have to define this target engine. This engine can be implemented in any technologies supported by the SCA Tools (Web service and WSDL, Java and so forth). All the other specific features of our composite will be automatically implemented by the extended SCA Tool.

Our goal is to provide a simple development platform for our composite service vision. We intend to take advantage of the SCA Tool simplicity and its diffusion in the Eclipse community. This tool will test our metamodel approach with some real application problems.
7. Toward the loose coupling notion

In this paper, our metamodel explicits all interlaced features involved in a composition of services and encapsulates them in a composite service. The arrangements between its architectural elements were the base toward the auto composition mechanism. This mechanism ensures some dynamic architectural modifications and their considerations by the composition logics. The auto composition intends to fulfill a main SOA criteria: maximizing the loose coupling in the composite architecture.

The notion of loose coupling is intuitively understood as a main objective of the SOA paradigm. The service orientation intends to maximise the reuses of existent software entities which are encapsulated by the concept of service. To automate these reuses, an important SOA challenge is the reduction of the dependencies: between collaborated services, and between these services and the wanted composites. This dependency reduction facilitates the composition of services. Moreover, it facilitates the replacement of services in this composition. However, this intuitive understanding of the loose coupling hides a lot of imprecisions in its definition. As for the composite service notion, we intend to fulfill this lack of theorizing. We want to provide a clear definition of the loose coupling and we want to define some measurement techniques in order to compare service composition approaches.

In fact, the existent SOA specifications [1,2,6] do not address the coupling explicitation and they rely on this intuitive definition. Moreover, researches on coupling in general focus on the definition of metrics which are only based on structural informations (methods, attributes and so forth [22]) or on execution informations (number of exchanged messages). There is a need of a loose coupling definition which takes account of all the SOA particularities: high level vision, abstraction from the implementation, service discovery, service selection, heterogeneities in the composition process and so forth.

Our definition of the loose coupling notion relies on our work about the composite service metamodel specification. In fact, we define our metamodel as the reification of all interlaced features involved in a service composition. Therefore, the concepts introduced in our approach are general and then, existent composition approaches can be matched in ours. These concepts will be used in our coupling specification and the derived metrics will allow approach comparisons.

Due to the length constraint, we only present an overview of our loose coupling specification which will be the subject of a dedicated paper.

Briefly, we define three different couplings in a composition of services which relies on the different phases of the service composition lifecycle [1]:

- first of all, an architect models his composite and the related service composition. He defines the required Fservices (or abstract services) and he defines their collaborations.
- Then, these informations are used by a discovery engine which proposes a set of candidate services, these are concrete services. A selection algorithm identifies the most suitable services.
- Finally, a composition engine establishes the collaborations between these concrete services, as a result a concrete implementation of the composite is defined.

These three phases are respectively linked to our three couplings: semantic coupling, syntactic coupling and physical coupling.

7.1. Semantic coupling

The semantic coupling focuses on the composite modeling. It represents the dependencies which involve the abstract services and the composite capabilities. The semantic coupling of an abstract service relies on the composite capabilities that it participates in. Thereby, the semantic coupling measurement of
the abstract services dependents on the collaboration schema (section 3.4) and on the level of importance given by the architect to the composite capabilities. We define three levels of semantic coupling:

- **Strong coupling**: an abstract service and a composite are strongly coupled if this service participates in an essential capability of this composite. A capability is subjectively identified as essential by the architect. According to his expertise, he semantically defines this capability as major for the composite: without this capability the composite becomes unusable.

- **Loose coupling**: an abstract service and a composite are loosely coupled if this service participates in a non essential capability of this composite. However, these capabilities have a direct impact on the composite efficiency. We can not guarantee the composite quality if one or more of these capabilities are removed. If all of them are removed, we rule that the composite becomes unusable.

- **Non predominant coupling**: an abstract service and a composite have a non predominant coupling if this service participates in a non essential capability. Moreover, all non predominant services can be removed without consequences on the composite availability or on the efficiency of its essential capabilities. These specific capabilities express some optional features.

### 7.2. Syntactic coupling

The syntactic coupling focuses on the dependencies between abstract services and concrete services. We define two levels of syntactic coupling:

- **Strong coupling**: an abstract service is strongly coupled with a concrete service if this concrete service is the only available service which can realize the abstract service. There are not optional solutions.

- **Loose coupling**: an abstract service is loosely coupled with a concrete service if there are optional solutions. The more there are suitable concrete services, the weaker the coupling is.

Thereby, this notion of syntactic coupling directly dependents on two elements:

- the used discovery algorithm: existent algorithm can be grouped in two approaches, *one to one matching* [26,27] and *one to many matching* [11,14]. One to one approach identifies one abstract service to exactly one concrete service. One to many approach identifies one abstract service to one or many concrete services in collaboration. Thereby, the one to many approach has a larger potential of solutions than the one to one. The syntactic coupling is weaker.

- the selection criteria: they define the constraints on the identification of concrete services. These constraints are provided to a discovery algorithm. Thereby, the weaker the constraints are, the larger the potential of solutions is. The management of the heterogeneities (section 3.3 and 2.5) directly impacts on the syntactic coupling. A composite which has not implementation considerations can use a larger range of concrete services.

### 7.3. Physical coupling

The physical coupling focuses on the concrete implementation of the composite. It reuses existent researches and it is based on empirical measurements [22,28] such as methods calls, message exchanges, the number of linked services, commune objects and so forth. These metrics allow for the identification of the physical dependencies between concrete services. They are fully linked to the collaboration methods (orchestration, choreography) and to the communication methods (messages, notification of events, and so forth). A purely SOA approach gives priority to the minimization of the transversal communications
between services in the same level of composition. There is a preference for a vertical communications 
(composite and constituent) than horizontal communications (constituent and constituent).

These three couplings intend to provide some clear comparison criteria. The different service com-
position approaches can be compared according the loose coupling notion. An architect will be able to 
select the most suitable approach.

Moreover, these couplings want to be a guideline for the composite service development. The semantic 
coupling allows for the localization of critical abstract services following the architect point of view and 
his expertise in the application domain of the composite. Therefore, he will focuses on decreasing the 
syntactic coupling associated with these abstract services. In fact, it is better to have numerous concrete 
services which can be used to realize a critical abstract service. In the same way, it is better to decrease 
the physical coupling of these concrete services, it will be easier to remove them from the composite 
arhitecture. All these precautions are used in order to minimize the risk of failures in the composite. 
The finality is to make these three couplings as weak as possible. The global coupling of a composite is 
the combination of these three couplings: the semantic, the syntactic and the physical.

8. Conclusion

In this paper, we presented an explicit composite service metamodel which provides a better un-
derstanding of the service composition. We pointed out the theoretical interest of this metamodel as a 
first step toward a consensus in the composite service definition in the SOA paradigm. We define this 
metamodel by reifying the main composition features at the architectural level. This approach allows us 
to specify the dependencies between these features in a single model. According to these dependencies, 
we define the auto composition process. This process ensures the remplacement of defective services. 
Our metamodel handles all the impacts on the architectural elements and on the composition logics.

Briefly, we pointed out the lack of precisions in the loose coupling definition. This notion is directly 
linked to the service composition. We show that the concepts of our metamodel can be used in the 
explicitation of the loose coupling notion. This explicitation allows for the definition of metrics. They 
will be used to compare available composition approaches and to assist an architect as development 
guidelines.

Our futur works will focus on different subjects. Firstly, we will finalize the second approach prototype 
in order to provide a tool. We intend to release our extended SCA Tools for the SCA community of Eclipse. 
This will be the opportunity to collect some feedbacks about our tool and its possible improvements.

Secondly, we will focus on our loose coupling vision and the publication of the results. These results will 
show a comparison between existent composition approaches. Moreover, the loose coupling definition 
will complete our composition toolbox. The architect will see on the fly measurements of his composite 
following our three different couplings. He will be able to visualize all the impacts of an architectural 
modification on the overall coupling of his composite. Therefore, our tool will propose some guidelines 
for the coupling minimization.

References


A novel QOS-based broadcasting scheme for wide area networks

B. Gupta\textsuperscript{a,∗}, N. Debnath\textsuperscript{b}, S. Rahimi\textsuperscript{a}, V. Videm\textsuperscript{a} and K. Ethirajan\textsuperscript{a}

\textsuperscript{a}Department of Computer Science, Southern Illinois University, Carbondale, IL, USA
\textsuperscript{b}Department of Computer Science, Winona State University, Winona, MN, USA

Abstract. In this paper we have proposed a practical approach for broadcasting in wide area networks. The concept of pseudo-diameter, an important parameter in the DVR data structure unused by the RPF method has been applied to prune the flow of packets in order to achieve reduction in the number of the duplicate packets generated during broadcasting. Performance comparison with the classical RPF method shows that the proposed method generates fewer duplicate packets and hence it is more band-width efficient. We have further enhanced our idea by incorporating a new pruning scheme to further reduce the number of duplicate packets; thereby even offering better QOS from the viewpoint of better bandwidth utilization.

Keywords: Broadcasting, reverse path forwarding, pseudo diameter

1. Introduction

Broadcasting refers to transmitting a message unconditionally to all the neighbors. A simple way of broadcasting is flooding. Flooding generates large number of duplicate packets [1] as it sends packet to every outgoing line; therefore it is not at all bandwidth efficient, particularly for large networks. In Spanning Tree based broadcasting, each router constructs minimal spanning tree [1]. But there is a major overhead to store the topological information of a network at each router. It is not efficient from the viewpoint of routers’ limited memory. This overhead increases while updating the topological information when there is a change in the topology. It makes this approach unpractical.

The algorithms used in modern day networking use the concept of either Distance Vector Routing (DVR) or Link State Routing (LSR). For broadcasting in Wide Area Networks (WAN) DVR [2,9] technique also known as Ford-Fulkerson algorithm [10] has been efficiently used in Reverse Path Forwarding (RPF) [3,4]. In LSR there is burden on the router memory as it needs to know the shortest path. RPF does not need to know the topology and this is a significant advantage over LSR. However, the main disadvantage of RPF is that it generates a reasonably large number of duplicate packets while broadcasting. This method has later been modified [5,6] to reduce the number of duplicate packets significantly. The method is known as Modified RPF (MRPF). It is a practical approach for broadcast routing in store-and-forward packet switching computer networks. MRPF retains all the advantages of the RPF method while it has the important additional advantage of efficient utilization of bandwidth by reducing further the number of duplicate packets generated. Thus it offers a better mechanism to control...
the duplicate packet generation than RPF. This approach introduced the notion of pseudo diameter which is the maximum delay present in the DVR table of a router. For different routers the contents of the respective DVR tables may vary, causing a possible variation in the magnitudes of the corresponding pseudo diameters as well. Therefore it is not the true diameter of the network and hence it has been termed as pseudo diameter. In this context, it may be mentioned that classical RPF method did not utilize the advantage that could be gained by the use of this information.

1.1. Problem formulation

The objective of our work is to design a broadcasting algorithm, different from the classical RPF method that utilizes the concept of pseudo diameter to reduce the number of possible duplicate packets generated when compared with the RPF method.

2. Pseudo diameter based pruning

In this section we will first explain the working principle of our approach with an example followed by a formal presentation of the algorithm.

A source node before broadcasting a packet, initiates the pseudo diameter field from its DVR table and neighboring node list in the packet. This list contains its neighbors such that each will receive a copy of the broadcast packet. An intermediate node before it forwards the packet to any of its neighbors verifies whether it has enough diameter to reach the neighboring node set and filters the nodes that had already received a packet from its sender. The filter process is done by comparing the neighbors list information received in the packet, with its own neighbors list. Diameter validation is done by comparing the pseudo diameter field in the packet with the delay required to reach that particular neighbor according to the DVR table of the forwarding node. If the forwarding node has enough pseudo diameters to forward a packet to its neighboring node, then before forwarding it first reduces the content of the pseudo diameter field in the packet by the delay to reach that neighboring node and then it forwards.

2.1. An example

Let us consider an example as shown in Fig. 1. In this figure, the delay between every two directly connected nodes (routers) is shown. Some of the DVR tables necessary for the explanation are shown.

Without any loss of generality, let A be the source of broadcast. Since A is the source it will fetch the maximum delay (10) in the DVR table and reduces the pseudo-diameter by its respective outgoing link
delay. Besides updated pseudo-diameter, source node piggybacks the neighboring node set information
< B, C, E > which are within the pseudo-diameter range and forwards a copy of the packet to each of
those links. The intermediate node C on receiving the packet, finds the nodes which are reachable
with the available pseudo-diameter (5). < D > is the node that is within the pseudo-diameter range.
From the piggybacked information C realizes that D did not receive the packet yet. Having enough
pseudo-diameter (5) to reach D (5), C forwards the packet to D by updating pseudo-diameter value to
‘0’. Meanwhile B on receiving the packet from A finds that the updated pseudo-diameter value in the
received packet is 6. With this it can reach < C, E >. From the piggybacked information < B, C, E > in the
packet, B realizes that < C, E > have already received a packet along with B. Hence B does not forward
the packet further. Thus B did not generate duplicate packets. In this way all nodes will receive a copy
of the broadcast packet. Note that some nodes may receive duplicate packets. We will show later that
the total number of duplicate packets is still less than RPF.

In the following theorems we denote the pseudo diameter of a broadcast source as \( \tau \).

**Theorem 1:** \[ \tau - (r_i + r_j + \ldots + r_m) \geq 0. \]

**Proof:** Let \( S_i \) be the source of broadcasting and \( r_i, r_j, \ldots, r_m \) be the respective reductions in the
pseudo-diameter (\( \tau \)) of \( S_i \) as the packet travels along a path consisting of the routers \( R_i, R_j, \ldots, R_m \).

In our approach, as a broadcast packet propagates, the value of \( \tau \) is reduced first at the source \( S_i \) by
an amount \( r_i \) which is equal to the delay to the next hop node \( R_j \) from \( S_i \), provided \( (\tau - r_i) \) is greater
than or equal to zero. This process of reduction continues at all intermediate nodes along the path to
\( R_m \) by the respective delays to their next hop nodes so long as the updated pseudo-diameter field of the
packet is greater than or equal to zero. Since pseudo diameter \( \tau \) implies that with \( \tau \) any node is always
reachable from \( S_i \), hence after all the reductions, the updated pseudo diameter field of the packet at any
intermediate node may contain either zero, if the delay from \( S_i \) to that node is \( \tau \); otherwise some positive
value. Therefore, the condition \[ \tau - (r_i + r_j + \ldots + r_m) \geq 0. \]

### 2.2. Algorithm broadcast

Input: Every node \( n_i \) maintains a set \( L_i \) of its neighbors and knows the delay to reach each neighbor.
This later information is present in the DVR table of node \( n_i \).

At Source \( n_s \):

sets \( \text{packet.Delay} = \text{pseudo diameter of node } n_s \); 
creates forwarding node set \( L_f^s = L_s - L_1^s \);

/*nodes within pseudo diameter range & \( L_1^s \) contains the nodes that are not within pseudo
diameter range of \( n_s \)*/
for each node $n_i$ in $L_s$
  /*source broadcasts to its neighbors*/
  loop
    get the delay to reach node $n_i$ into $\tau^i$;
    sets packet.src_addr = $A_s$
    /*store source address in packet*/
    sets packet.Delay = packet.Delay $-$ $\tau^i$;
    /*update pseudo diameter*/
    sends the packet[$A_s$, packet.Delay, $L_f$] to $n_i$;
  end loop
end for

At an intermediate node $n_i$ receiving a packet from $n_j$, $j \neq i$:
receives a packet;
creates forwarding node set $L_f^i = [L_i - L_f^j] - [L_i \cap L_f^j]$;
/*identify the neighbors within pseudo diameter range of $n_i$, out of which nodes that have received a packet from $n_j$ are pruned*/
for each neighbor node $n_k$ in $L_f^i$, $k \neq i$
  /*intermediate node broadcasts to neighbors*/
  loop
    get the delay to reach node $n_k$ into $\tau^k$;
    sets packet.Delay = packet.Delay $-$ $\tau^k$;
    /*update the pseudo diameter*/
    sends the packet [$A_i$, packet.Delay, $L_f^i + [L_i \cap L_f^j]$] to neighbor $n_k$;
  end loop
end for

Theorem 2: Every node receives a copy of the broadcast packet.
Proof follows directly from Theorem 1.

3. Performance comparison

The proposed pseudo diameter based pruning generates fewer duplicate packets while broadcasting compared to Reverse Path Forwarding (RPF). It results in better utilization of network bandwidth.

3.1. Simulation results

Simulation experiments were conducted to compare our approach with Reverse Path Forwarding from the viewpoint of the number of duplicate packets generated during broadcasting. In this simulation we have used three different network topologies with 10, 15, 20, 25, 30 nodes respectively. In each topology nodes have been randomly placed and links have been associated with randomly chosen costs (delays). For each node set we have performed the following experiment using both our algorithm and RPF.

In each topology of a node set (e.g. topology 1 of 25 node set) we have repeated the experiment considering each node as a source and recorded the average number of packets generated ($X_1$). Similarly
we have recorded the average number of packets generated ($X_2$, $X_3$) for the other two topologies for the same node set. Finally average of $X_1$, $X_2$, and $X_3$ is the average number of packets generated for a node set. The results of the experiments are shown in Tables 4(a–f). We have used these final average values for the different node sets to compare the performance as shown in Fig. 2.

Figure 2 clearly shows the advantages, our algorithm (specified as ALG in the figure) offers over RPF. The simulation results confirm that our approach requires less number of packets for the broadcasting process. This indicates that our proposed approach is a better choice than RPF from the viewpoint of offered quality of service which in this case is the efficient utilization of bandwidth.
4. Modified pseudo diameter based pruning

The idea of pseudo diameter when combined with piggybacking the forwarding node list paved way for pseudo diameter based pruning [8]. It is superior to RPF from the view point of fewer number of duplicate packets generated during broadcasting. In this section we further propose an enhancement of the idea used in Algorithm Broadcast to make it suitable for broadcasting with further reduced number of duplicate packets. To achieve it, we will introduce a scheme known as modified pseudo diameter based pruning. We will first explain the working principle of our approach with an example followed by a formal presentation of the algorithm.

A source node before broadcasting a packet, initiates the pseudo diameter field from its DVR table and neighboring node list in the packet. This list contains its neighbors such that each will receive a copy of the broadcast packet. An intermediate node before it forwards a packet checks whether the pseudo diameter received in the packet is less than the pseudo diameter of itself. If it is greater, then the pseudo diameter in the packet is replaced with the pseudo diameter of the intermediate node. The intermediate node also, prunes some of its neighbors which have already received a packet from its sender and verifies whether it has enough diameter to reach the rest of neighboring node set. The filter process is done by comparing the neighbor list information received in the packet, with its own neighbors list. Diameter validation is done by comparing the pseudo diameter field in the packet with the delay required to reach that particular neighbor according to the DVR table of the forwarding node. If the forwarding node has enough pseudo diameter to forward a packet to its neighboring node, then before forwarding it first reduces the content of the pseudo diameter field in the packet by the delay to reach that neighboring node and then forwards.

4.1. An example

Let us consider the network topology as shown in Fig. 3 to illustrate the idea. Ignore for the time being the delays present in parenthesis. Let Node B be the broadcasting source. Source node from its own DVR table gets the pseudo diameter which is the maximum of all the delays present in its DVR table. The pseudo diameter of ‘B’ is 60 (maximum delay of node B from Table 6). Node B checks whether its pseudo diameter is greater than its outgoing link delays. If so node B reduces the pseudo diameter by its outgoing link delay and forwards on to BA, BC, BE. In Fig. 4 these links are indicated by solid lines.

In the forwarded packet, Node B also piggybacks the Node set <A, C, E> to which it has forwarded. Node A receives a packet from B with pseudo diameter 20 (60–40). It then compares the received pseudo diameter with the maximum delay present in its DVR table which is 60 from Table 5. The pseudo diameter will not be updated because the maximum delay present in its DVR table is not less than the received pseudo diameter (60 > 20). From the piggybacked information present in the packet, A finds
that among its neighbors F is the node which is in pseudo-diameter range and which has not yet received a packet. After this diameter validation node A reduces the pseudo diameter by the outgoing link delay and forwards on to the link AF. Node F accepts the packet and does not forward further because its received pseudo diameter is 0. Similarly Node E received a broadcast packet with pseudo diameter 0 from B. So it cannot forward further.

Let us focus on node C where we can notice the difference between Pseudo Diameter Based Pruning and Modified Pseudo Diameter Based Pruning after there is a change in delay. Let us now assume that the delays between CA and CE have reduced to 30 each and the delay FE has reduced to 10 which are shown in the parenthesis (Fig. 3). In Pseudo Diameter Based Pruning node C reduces the received pseudo diameter by its outgoing link delays and forwards on to links CD and CF. But whereas in the modified approach node C compares the received pseudo diameter with the maximum delay present in its DVR table. If the maximum delay present in its DVR table is less than the received pseudo diameter, then node C changes its pseudo diameter with the maximum delay present in its DVR table.
Fig. 5. Modified pseudo diameter based pruning execution.

From Fig. 3 after there is a change in delays node C can reach A and E with lower delays than the actual delays present in its DVR table. In our approach as shown in Fig. 5, node C again runs the DVR algorithm and calculates its new DVR table with the changed delays. Now the maximum delay of C is 30 from Table 4 to reach nodes A and E directly. The updates from node C will take some time to reach other nodes. In the meantime when C receives broadcast packet from B with the old pseudo diameter which is 40(60–20), it now compares this received pseudo diameter with the maximum delay present in its new DVR table which is 30. The maximum delay present in the DVR table is less than the received pseudo diameter. So node C changes its received pseudo diameter with the maximum delay present in its new DVR table (30). From the piggybacked information C received <A, C, E>, it finds that D and F are the nodes that did not receive a packet yet and are in the pseudo-diameter range. So Node C forwards the packet to nodes D and F. Node D cannot forward the packet further since the received pseudo-diameter is 0. From the piggybacked information F received <A, C, E, D, F> it finds that among its neighbors G is the node that has not yet received a packet and also it is in the pseudo-diameter range. So F forwards a packet to G. Node G cannot forward further as its received pseudo-diameter is 0.

From Figs 4 and 5 we observe how the modified approach has reduced number of duplicate packets when compared to Pseudo diameter based pruning. In both approaches node F sends the packet to G since it has enough pseudo diameter to reach node G. So node G in both approaches accepts the packet from F. In Pseudo diameter based pruning method node G again receives the broadcast packet from node D via link DG since node D has enough pseudo diameter to reach G. Whereas in our approach node D does not send the packet via DG since its received pseudo-diameter is 0. This is because node C changed its received pseudo diameter with the maximum delay present in its DVR table which is 30 which is not true for pseudo diameter based pruning method. Thus the modified approach reduces the extra duplicate packet that is generated when compared to pseudo diameter based pruning method. The above discussion leads to the following observation stated below.
**Theorem 3**: Number of duplicate packets in Modified Pseudo Diameter Based Pruning \(\leq\) Number of duplicate packets in Pseudo Diameter Based Pruning.

### 4.2. Algorithm modified-broadcast

Input: Every node \(N_i\) maintains a Set \(L_i\) of its neighbors and knows the delay to reach each neighbor. This later information is present in the DVR table of node \(N_i\).

**At Source \(N_s\):**

sets packet.Delay = pseudo diameter of node \(N_s\);
creates forwarding set \(L_f^s = L_s - L_1^s\);

/*nodes within pseudo diameter range \& \(L_1^s\) contains the nodes that are not within pseudo diameter range of \(N_s\)*/
for each node \(N_i\) in \(L_f^s\)

/*source broadcasts to its neighbors*/
loop
get the delay to reach node \(N_i\) into \(\tau_i\);
sets packet.src.addr = \(A_s\)

/*store source address in packet*/
sets packet.Delay = packet.Delay – \(\tau_i\);
/*update pseudo diameter*/

sends the packet \([A_s, packet.Delay, L_f^s]\) to \(N_i\);
end loop

**At an intermediate node \(N_i\) receiving a packet from \(N_j, j \neq i\):**

receives a packet;
creates forwarding node set \(L_f^i = [L_i - L_1^i] - [L_i \cap L_f^j]\);
/*identify the neighbors within pseudo diameter range of \(N_i\), out of which nodes that have received a packet from \(N_j\) are pruned*/
If pseudo diameter \((N_i) < packet.Delay\)
sets packet.Delay = pseudo diameter \((N_i)\)
end-if
/* obtain the minimum delay to reach all other nodes*/
for each neighbor node \(N_k\) in \(L_f^i\), \(k \neq i\)
/*intermediate node broadcasts to neighbors*/
loop
get the delay to reach node \(N_k\) into \(\tau_k\);
sets packet.Delay = packet.Delay – \(\tau_k\);
/*update the pseudo diameter*/
sends the packet \([A_i, packet.Delay, L_f^i + [L_i \cap L_f^j]\) to \(N_k\);
5. Performance comparison

The proposed modified pseudo diameter based pruning generates fewer duplicate packets while broadcasting compared to pseudo diameter based pruning. It results in better utilization of network bandwidth.

5.1. Simulation results

Simulation experiments were conducted to compare the two proposed algorithms in this paper and RPF from the viewpoint of the number of duplicate packets generated during broadcasting. As described in Section 3, here also in the simulation we have used three different network topologies each with 10, 15, 20, 25, 30 nodes respectively. In each topology nodes have been randomly placed and links have been associated with randomly chosen costs (delays). Experiments as in Section 3 have been performed for all node sets using all three approaches. Table 9 shows the average number of packets generated by the three approaches for the different node sets.

Figure 6 shows the comparison of the average number of broadcast packets generated by RPF, Algorithm Broadcast, and Algorithm Modified-Broadcast. It is clear that Algorithm Modified-Broadcast shows far superior result compared to RPF. It also shows improvement over Algorithm Broadcast.

6. Conclusions

In this work we have utilized some important information termed as pseudo diameter to reduce the number of duplicate packets generated during broadcasting in WANs. This information although is present in distance vector routing tables, yet is not used by classical reverse path forwarding method for broadcasting purpose, which eventually uses these DVR tables as its main data structures. We have shown how pseudo diameter helps in the reduction of duplicate packets compared to RPF and hence offers better QOS from the viewpoint of bandwidth utilization.
We have also proposed an enhancement of the idea used in Algorithm Broadcast to design another algorithm, Algorithm Modified-Broadcast that guarantees further reduction in the number of duplicate packets. To achieve it, we have introduced a new scheme known as modified pseudo diameter based pruning. It offers even better QOS from the viewpoint of better bandwidth utilization when compared to Algorithm Broadcast.

References

Enabling tool reuse and interoperability through model-driven engineering

Zekai Demirezen\textsuperscript{a,∗}, Yu Sun\textsuperscript{a,∗}, Jeff Gray\textsuperscript{b} and Frédéric Jouault\textsuperscript{c}
\textsuperscript{a}Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL, USA
\textsuperscript{b}Department of Computer Science, University of Alabama, Tuscaloosa, AL, USA
\textsuperscript{c}AtlanMod (INRIA & EMN) Nantes, France

Abstract. Software components provide a wide range of functionality that can be used across several domains. In some cases, reuse at a very coarse level of granularity (e.g., reusing functionality provided within an existing tool) is desirable, but challenging to realize due to the interface boundaries of the tool and the unanticipated level of reuse. This paper describes our results in applying model-driven engineering (e.g., domain-specific modeling and model transformation) to the tool reuse problem. Our approach captures the essence of each tool in a metamodel and uses model transformations to map between the tool representations. Specifically, we describe our results in reusing the graphical layout functionality provided by one tool (e.g., GraphViz) that does not exist natively in another tool (e.g., the Eclipse Graphical Modeling Framework).

Keywords: Domain-specific modeling, tool interoperability, model transformation

1. Introduction

Software-based tools provide a wide range of functionality in the context of their domain. However, in some situations a tool may lack a specific desirable operation needed by a user. Other tools in the same or even different domain may include these missing operations, or may provide an improved implementation of the specific desired functionality in a different context. This paper describes our results in reusing the functionality of an open source tool called GraphViz\textsuperscript{1} to embed the layout functionality that is needed in Eclipse's Graphical Modeling Framework (GMF).\textsuperscript{2} We implemented an Eclipse plug-in that enables the execution of the GraphViz Layout algorithm during modeling operations in the GMF editors. The main challenge we encountered was focused on the issue of data interchange between the tools. Because both GraphViz and GMF operate on their own unique data structures, we designed an exchange mechanism to adapt their diverse formats. The specific solution to these challenges involved the application of techniques from model-driven engineering (MDE), such as domain-specific modeling (DSM) \cite{1} and model transformation \cite{2}, to specify each tool format and to convert between the representations.

The sharing of information among similar tools enables intellectual assets to form a homogenized collection of information. Although the topic of tool interoperability is important, there are other cases

\textsuperscript{∗}Corresponding author. E-mail: {zekzek,yusun}@cis.uab.edu.
\textsuperscript{1}http://www.graphviz.org/.
\textsuperscript{2}http://www.eclipse.org/modeling/gmf/.

1472-7978/10/$27.50 © 2010 – IOS Press and the authors. All rights reserved
of interoperability that can yield additional benefit. In particular, the concept of sharing functionality (as an alternative to sharing just data) between tools can enable a tool chain whereby features not available in one tool can be realized through collaboration with other tools. The challenge of such sharing of functionality is generally twofold: 1) transforming the representation of a source tool into that expected by the target tool providing the desired functionality; 2) transforming the result back to the first tool (or possibly to a third one). This is challenging because reuse at such a coarse level of granularity transcends the boundaries of the tools in a way that was likely not anticipated when each tool was separately designed.

We utilized DSM and model transformation to overcome interoperability issues between GraphViz and GMF, which provided an adaptable and maintainable solution that minimized tool adaptation effort. DSM is an innovative software engineering approach to decouple the description of the essential characteristics of a problem from the details of a specific solution space. A Domain-Specific Modeling Language (DSML) can be utilized to declaratively define a software system using the specific domain concepts. A DSML is specified by a metamodel, representing abstract syntax that describes domain concepts and their relations, to which concrete models can be constructed that conform to the definition of the metamodel (e.g., similar to the way that a program conforms to the grammar of its defining language). Models can then be used to automatically generate the desired software artifacts (e.g., programming code, simulation scripts, XML deployment description) using model transformation engines and generators.

Model transformation, which is a core technology of MDE and complements DSM, receives a source model in a certain domain as input, and produces as output another model in a given target domain. If the various tools are considered as different domains, model transformation can be applied to transform the tool artifacts (i.e., models) between different tools so that interoperability can be realized. Our experience in applying these ideas has shown effective results using this approach. There are three main reasons that contributed to this reduction in effort:

1. In this specific context, both the source and final target information were already expressed as models. For example, the GMF metamodel already exists in the format we needed for this integration process.
2. There is available tool support to translate between models and context-free syntax, such as the intermediate format corresponding to the input and output of the GraphViz tool. However, there is the additional cost of specifying this syntax in an appropriate formalism.
3. Moreover, our choice of domain-specific modeling and model transformation for this task is also motivated by the modularity and reusability such an approach provides. For instance, the GraphViz syntax specification may be reused in other contexts, such as providing graph layout capabilities to editors other than GMF. Our approach makes this possible because the syntax specification is completely decoupled from the mapping rules.

The basic idea of the model-driven solution is to define the abstract syntax (represented as a metamodel) and the concrete syntax (i.e., the manner in which a diagram is visualized) for GraphViz and GMF, and then execute associated transformations to map data between the tools. Figure 1 shows the data flow of exporting an unorganized design model from GMF to GraphViz, and importing the results of the arrangement suggested by the layout algorithm back into GMF.

After first presenting the related work in Section 2, we then introduce and define metamodels for the two tools in Section 3. The detailed implementation of interoperability between these two tools is given in Section 4, with a brief introduction about the AmmA platform that we used to realize the whole process. In Section 5, we conclude the paper and present some lessons learned from this experiment.
2. Related work

In tool interoperability, data exchange is accomplished over well-defined data structures and exchange formats [6]. However, different standards and formats make software interoperability a challenge. There are several existing approaches that can be adopted to realize tool interoperability. For example, a general approach that is based on traditional parsing and interpreting activities can be implemented with general-purpose programming languages (e.g., Java, C#) [3]. In addition, XML-based interoperability has emerged as a popular choice for a generic exchange format among software tools [4,5]. However, although this works relatively well when all considered tools use some form of XML, it is not a convenient solution when applied in other contexts that involve legacy tools (e.g., when context-free parsing is necessary to read from a proprietary file format). A new approach to interoperability, which is advocated in this paper, is based on domain-specific modeling and model transformation techniques, where the different formats are captured as: 1) abstract definitions of data structures (i.e., metamodels)
and 2) concrete syntax definitions (e.g., context-free, XML), and finally 3) transformation rules mapping from one representation to another.

The majority of tool interoperability research has focused on the data exchange aspect of integration [7–9]. In fact, we recently investigated this type of tool interoperability by applying domain-specific modeling to enable sharing of information between a set of code clone tools [10]. Crnkovic et al. [11] proposed a generic framework, named DUALLY, to allow transformation among different component models. In DUALLY, structural aspects of interoperability are covered, but behavior models are not addressed. The framework is provided as an Eclipse plug-in. Kappel et al. [12] discussed semantic integration of modeling languages. They demonstrated a framework that utilizes ontologies and semantic links to generate model transformations. Garlan et al. [13] studied architectural description technologies to address the interoperability problem in component-based systems. Their work provides a platform for architectural data interchange. Jonkers et al. [14] utilize model transformation to exchange between architecture description language formats. Fabro and Valduriez [15] focus on generation of weaving models from structural links between meta-models. These weaving models are created from similarity analysis of matching transformations.

Kuhn and Murzek [16] investigated business process model interoperability within the Business Process Management toolkit called ADONIS. They introduced an approach using Enterprise Application Integration (EAI) to model the concepts of workflow integration in heterogeneous systems. They developed the model transformer tool that supports transformation of source models to the ADONIS configurable metamodels. Duddy et al. [17] introduced an XSLT-based transformation tool to convert Meta-Object Facility (MOF) metamodels in XML Metadata Interchange (XMI) format to Eclipse Modeling Framework (EMF) metamodels. Clark et al. [18] proposed a metamodeling environment that enables language definition and metamodel mapping. Their environment consists of XMap and XSync transformation languages. Blanc et al. [19] introduced an architecture for addressing data exchange between services. Their prototype enables the automation of service connection by middleware technologies, such as CORBA and Web Services.

Kim et al. [20] discussed model-driven data integration and demonstrated how to merge data from different models. They defined relationships between model elements at the conceptual level. Kern and Kuhne [21] addressed model interchange between ARIS (ARchitecture of integrated Information Systems) and the EMF. They provided concept mappings between the domains at the metamodel level and meta-metamodel level (i.e., the metamodel defining a metamodel). Siegmund et al. [22] utilized feature-oriented programming to describe the runtime adaptation of services. They showed how software product line techniques can be used to enable collaboration between systems based on service-oriented architectures. Bonde et al. [23] introduced an approach based on traceability models. They showed how to keep links between source and target model elements. They only considered two systems that were derived from one platform-independent model. Staub et al. [24] explained the four steps of their approach to conceptual data modeling. They solved interoperability issues using higher levels of abstraction. Data format transformation can be deduced from the model mapping automatically.

The main limitations in the above related works are: 1) they lack a systematic approach to define a certain tool or data format, including the parsing and storage of the data if proprietary formats are used in legacy tools; 2) some low-level implementation details cannot be avoided during the data exchange or transformation process; 3) they often do not provide support to define explicitly the semantics of complex kinds of mappings (e.g., mapping expressions where multiple attributes in a source model combine to form a single model in the target, or vice versa); 4) the extensibility and reusability of most approaches is not adequate, with many concerns tangled across the interoperability solution. In the next
section we introduce the manner in which we capture the essence of each software tool using DSM, with Section 4 providing the details on how we use model transformation to perform the mapping between the representations.

3. Domain definition and tool metamodeling

This section provides an overview of the representation formats of GraphViz and GMF, which are the two tools that we considered as case studies in our interoperability example. The metamodel for each tool is specified in this section, with the corresponding mappings described in Section 4 as model transformations.

3.1. Layout DSL in GraphViz

GraphViz is an open-source graph visualization tool that has several layout programs for the placement of nodes and edges in graphs. One of the layout programs of GraphViz is the Dot Layout tool, which reads directed graphs, computes layouts, and produces an output of attributed graphs which includes layout coordinates [25]. Figures 2 and 3 show a sample Dot file and the layout of this graph. Before the execution of the auto-arrange algorithm, the Dot file includes only node and edge definitions without position attributes (top of Fig. 2). All the position data are inserted after the execution of the layout algorithm. For example, in the “Before Layout” fragment, the content -> chapter and content ->

![Fig. 2. Sample graph with position attributes.](image-url)
appendix statements define two edges between a content node and the chapter, appendix nodes. The “After Layout Fragment” (bottom of Fig. 2) is an excerpt from the definition of the graph as re-serialized by Dot. The excerpt only shows the content node and its edges (all other nodes are removed for space consideration) with generated attributes (i.e., pos, width, and height). In this fragment the content [pos="185,162", width="0.92", height="0.50"] portion specifies position values of the content node within the graph. Other parts correspond to the definitions of edges linked to the content node and their position values. In essence, the changes between the “before” and “after” version of this graph representation correspond to the specific functionality that we desire to reuse in GraphViz (i.e., positional layout restructuring based on coordinates determined by Dot).

To enable the reuse of the GraphViz functionality, the abstract representation of GraphViz is specified as a metamodel. Figure 4 shows the Dot metamodel, which we inferred by reading the user manual of Dot. The primary concept in Dot is a Graph consisting of Graph Elements that can be Nodes or Edges. Both of these kinds of elements can have attributes such as shape, label, position, width and height. The Attribute meta-element represents these attribute values.

### 3.2. Representation of GMF concepts

The EMF\(^3\) assists in building applications that are described by a data model that supports code generation. EMF provides tools and runtime support for manipulating models to produce a set of Java classes for the model. The meta-metamodel of EMF is Ecore. A subset of Ecore is shown in Fig. 5. The Graphical Editing Framework (GEF)\(^4\) can be used to design a graphical editor for a specific data model. The GMF provides a supporting infrastructure for developing graphical editors based on EMF and GEF. Models in GMF are defined by an Ecore metamodel.

---
\(^3\)http://www.eclipse.org/modeling/emf/.
\(^4\)http://www.eclipse.org/gef/overview.html.
4. ATL transformations supporting tool reuse

We defined the required metamodels and implemented model transformations using the AtlanMod Model Management Architecture (AmmA) [27], which enabled us to accomplish all modeling steps in a single modeling environment. A brief introduction to AmmA and its key components is given in this section followed by our AmmA-based solution.

4.1. Overview of AmmA

AmmA is a model engineering framework that may be used to capture DSLs as coordinated sets of models. The work described in this paper uses three of the main capabilities of AmmA: metamodeling,
model transformation, and projections to and from other technologies (e.g., grammars, XML) to further explore model-driven tool interoperability.

4.1.1. Metamodeling in KM3

The Kernel MetaMetaModel (KM3) [28] is the metamodeling language of AMMA, which enables the specification of metamodels. KM3 is comparable to the metamodeling language proposed by the Object Management Group (OMG), called the Meta Object Facility (MOF). KM3 can also be compared in purpose to Ecore, which is used in EMF. Metamodels defined in KM3 may be used directly, or be first transformed into their MOF or Ecore equivalents. KM3 also offers the possibility to define metamodels using a simple textual syntax, of which several examples are given throughout this paper. KM3 uses classes to capture concepts and references to capture relations between them. Classes may have attributes to represent specific properties (e.g., name, dimension, or value). A pair of opposite references can form a bi-directional association (i.e., setting one reference automatically sets its opposite). Additionally, some references may represent containment of other modeling entities.

4.1.2. Transformations in ATL

After the abstract syntax for different languages or tools is defined, it is often necessary to transform a source model into a separate target model. This is typically what has to be performed in tool interoperability scenarios. Model transformation enables the specification of executable mappings between metamodels. AMMA provides ATL (Atlas Transformation Language) [29] to specify transformations declaratively. Rules are used to specify the kind of target elements that have to be created for each of the kinds of source elements. These rules also specify how the properties of the target elements are initialized using values retrieved by navigating the source model.

4.1.3. Projections: Injections and Extractions

Metamodeling and model transformation are two core model engineering techniques. However, they are not enough to deal with concrete cases, because source and target models are often not specified as abstract graphs, but as concrete tools using specific data formats (e.g., based on grammars or on XML schemas). AmmA offers several tools to deal with concrete syntax, which are identified as projectors. Such a tool performs projections, which are two special kinds of transformations: 1) an injection produces a model from a concrete file; 2) an extraction takes a model as source and creates a concrete file as target. One such tool provided in AmmA is TCS (Textual Concrete Syntax) [30], which enables the specification of context-free concrete syntax. After a TCS model has been specified, it is possible to inject programs as models, and to extract models as programs.

4.2. Representing abstract and concrete syntax

At the beginning of our project we defined metamodels for each tool (shown earlier in Figs 4 and 5 as UML models, but their specific definition is in KM3). An existing GMF metamodel definition allowed us to reuse this definition as input to ATL transformations without further specification, because ATL is compatible with Ecore. However, we required the Dot metamodel definition in KM3 to provide a target for the required transformations. Figure 6 shows the KM3 specification of GraphViz Dot.

4.3. Defining model transformations for reuse

In AmmA, transformation operations require models to be loaded before the transformation executes. Because Dot files have a specific format, there is a need to parse and load them into models that conform
abstract class LocatedElement {
    attribute location[0-1] : String;
    attribute commentsBefore[*] ordered : String;
    attribute commentsAfter[*] ordered : String;
}
class Graph extends LocatedElement {
    attribute isConnected : Boolean;
    attribute name : String;
    reference elements[*] container : GraphElement oppositeOf owner;
}
abstract class GraphElement extends LocatedElement {
    reference owner : Graph oppositeOf elements;
}
class Node extends GraphElement {
    attribute name : String;
    reference attr_list[*] container : A_Item_List;
}
class Edge extends GraphElement {
    reference left : Node;
    reference right : Node;
    attribute isDirected : Boolean;
    reference attr_list[*] container : A_Item_List;
}
abstract class Attr_Stmt extends GraphElement {
    reference attr_list[*] container : A_Item_List;
}
class Node_Attr_Stmt extends Attr_Stmt{
}
class Graph_Attr_Stmt extends Attr_Stmt{
}
class Edge_Attr_Stmt extends Attr_Stmt{
}
class A_Item_List extends LocatedElement {
    reference a_item_list[*] container:A_Item;
}
class A_Item extends LocatedElement {
    attribute leftID: String;
    attribute rightID[0-1]: String;
}

Fig. 6. GraphViz Dot KM3.

to the metamodel of Fig. 6. TCS injection and extraction methods facilitate these activities. We defined the concrete syntax of the Dot format in TCS. An excerpt from the TCS definition of Dot is shown in Fig. 7. Each template provides the concrete syntax specifications of a meta-element. For example, the Graph template enables Graph attributes to be parsed and also pretty-printed (i.e., TCS defines not only the mappings between model elements and the textual representations, but also the exact format of the text). Graph elements are listed in a Dot file between the brackets and are separated by semi-colons. Please note that the concepts defined in the KM3 metamodel (such as Attr_Stmt and A_Item in Fig. 6) appear in the syntax definition of TCS (in Fig. 7).

We defined two transformation specifications for this case study (GMF2Dot.atl and MergeDotPositionIntoGMF.atl). Figure 8 shows the GMF2Dot definition that specifies the EMF Ecore (visually shown in Fig. 5) to Dot (visually shown Fig. 4) meta-element mappings.
Each model transformation rule defines an element mapping from a source metamodel to an element of a target metamodel. For example, the Node2Node rule provides a transformation between Ecore Node elements and Dot Node elements. In this rule, Node type corresponds to the Class Node in the GMF metamodel. The Edge2Edge rule is accomplished by checking the edge type because of the fact that the association Edge is an Edge in the GMF metamodel, which has type AssociationLink. Finally, edge source and target meta-elements, which are Node elements, are mapped to Dot Node elements to complete the “Ecore Edge to Dot Edge” transformation.

Figure 9 shows an excerpt of the MergeDotPositionIntoGMF transformation. This transformation is a merging activity, which combines a Dot and GMF model into a refining model to generate the resulting GMF model. Imperative statements are implemented to copy elements in the entrypoint rule Main (an entrypoint rule is the initial rule that initiates a transformation). During the transformation, generated position values from Dot (e.g., x, y, width and height attributes) are assigned into corresponding node elements of GMF attributes. In this rule, the GMF!Bounds helper function is used to query position information from the Dot model.
helper context GMFBounds

def: dotPosition : TupleType (x: String, y: String, w: String, h: String) =
let data: Set(DotNode) = DotGraph.allInstances()
->first().elements
->select(e | eoclIsTypeOf(DotNode))
->select(e | e.name = self.refImmediateComposite().element.name)
in
let data2: Set(DotA_item) = data->first().attr_list->first().a_item_list
in
Tuple{
  x = data2->select(e | e.leftID = "pos").first().rightID->first(),
  y = data2->select(e | e.leftID = "pos").first().rightID->last(),
  w = data2->any(e | e.leftID = "width").rightID,
  h = data2->select(e | e.leftID = "height").first().rightID
}

entrypoint rule Main() {
  do {
    for(b in GMFBounds.allInstances()) {
      b.x <- b.dotPosition.x;
      b.y <- b.dotPosition.y;
      b.width <- (b.dotPosition.w.toReal());
      b.height <- (b.dotPosition.h.toReal());
    }
  }
}

Fig. 9. Excerpt of the MergeDotPositionIntoGMF.atl.

4.4. Overview of transformation process

Figure 10 shows the complete GMF to Dot transformation scenario. The GMF metamodel defines GMF models and the Dot metamodel defines GraphViz models. To summarize, the steps of the required transformations are as follows:

- Defining a merge transformation between the source and output models (arranged model) using ATL (MergeDotPositionIntoGMF.atl)
- Defining the Dot metamodel (Dot.km3) (Note: GMF metamodel already exists)
- Defining the Dot concrete syntax (Dot.tcs)
  * Required to extract Dot models in AmmA to Dot models in GraphViz
  * Used to inject Dot models in GraphViz to Dot models in AmmA after the execution of auto layout
- Defining the transformation of GMF models to Dot models with ATL (GMF2Dot.atl)

The GMF model is transformed using GMF2Dot.atl. The generated Dot model is extracted and manipulated in Graphviz. The Dot.exe program, which is in the Graphviz tool, is executed to arrange the model. After this operation, the manipulated model needs to be transferred back into the GMF model. AmmA defines a set of injectors/extractors enabling the import/export of models from/to different domains. For example, a Dot model is transformed to GMF model with an injector. Finally, the model that has been auto-arranged and the existing model are merged to produce the final GMF model (using MergeDotPositionIntoGMF.atl).

Imperative statements are implemented as mappings that translate elements between the source and target models. During the copy, generated position values are assigned to corresponding node elements of GMF. At the end of these steps, a GMF model has been auto-arranged using Dot algorithms. With ATL,
by defining formal transformation rules, this case study has demonstrated that an algorithm available from the Dot tool can be used in the GMF tool, thus achieving interoperability at the functional level (not merely data exchange). Details (including a video demonstration) about the complete implementation described here is available at our project website at: http://www.cis.uab.edu/zekzek/gmf-dot/.

An Eclipse plug-in was implemented to provide easy access to graph arrangement within Eclipse Editors. The plug-in executes all ATL transformation steps via ANT tasks. The ANT tasks consist of loading metamodels and models, executing the Dot program, transforming models and saving models. Figure 11 shows the execution of the plug-in. A user can access the plug-in within the Eclipse editors by the Graphviz AutoLayout sub-menu, which invokes the ANT tasks and updates the Eclipse editor to show the arranged model.

5. Conclusion

At the beginning of this research we knew that it would be difficult to reuse the functionality of tools like GraphViz due to the coarse grained nature of such reuse. Because such tools operate as stand-alone applications, we needed to understand their interfaces and assumptions to enable the reuse of functionality. Domain-specific modeling, coupled with model transformation, enabled us to focus on the clear and organized mapping structure between meta-elements of the tools. The declarative specification of models and transformation rules were the primary representation that enabled the reuse of tool implementation, rather than implementing conversion tools in a general-purpose language. This
raised the abstraction of the problem description to a more manageable level. There are several lessons that were learned from applying model transformation to the tool interoperability problem (as noted in an earlier report [10]):

- **Model transformation provides separation of concerns across the integration process**: Separation of concerns in our experiments is shown in four areas: 1) The whole process is composed of three parts – defining source domain, defining target domain, and writing transformation rules; 2) When defining a domain, the metamodel (abstract syntax) and parser (concrete syntax) are specified separately. The AMMA platform connects the abstract syntax and concrete syntax when performing projections; 3) Metamodels have a clear and organized structure capturing the components and relationships they contain, which enables users to focus more on the concept and semantic mappings when writing transformation rules.

- **Adaptability and extensibility in defining new tools**: It is often necessary to define a new tool or DSL when implementing tool interoperability. We observed that a certain level of modularity may be achieved by introducing intermediate (or pivot) metamodels in the transformation chain between the source and target domains. For instance, a generic or pivot tool is often needed to optimize the
exchange among different tools. In the context of model transformation, only a metamodel is needed for a new tool. Although some tools with complex functions may make the metamodel difficult to specify, we can define a subset of the tool to satisfy the specific need of a tool interoperability case. In some other cases, a parser (concrete syntax) is also needed to support a new tool. In this situation, the extensibility depends on the complexity of the grammar, because building a parser is not an easy task [31]. However, because most of the tools are domain-specific, their grammar and parser are generally much simpler than general-purpose programming languages. In addition, some preprocessing can simplify the complex detection report text in order to make the parser easier to build. Admittedly, if the tool or data has a very complex syntax that makes building the parser extremely difficult, additional effort is needed.

- **Models should be the primary representation for tools:** Models are not the initial and final representations for most tools, so an extra initial effort is needed to inject and extract the tool data into a model. This involves multiple steps, and might make the whole approach less direct and efficient than source-to-source transformation. Fortunately, some powerful facilities are provided to simplify the exchange between sources and models. For example, in AmmA, XML can be automatically generated from the XML model by an embedded XML engine.

Although this paper is focused on the interoperability of two very specific tools (i.e., GMF and GraphViz), the general principles of metamodeling and model transformation are applicable to many different domains and tool integration efforts. Because the realization of interoperability also depends on the data used by each tool, parsing the data to inject into a model is an indispensable step for model transformation to enable tool interoperability. However, not every tool provides a structured data format that can be parsed. Some tools may not have an explicit data model that can be used; some others may offer a special format that is not even context-free, which is challenging to process. Therefore, how to parse and process these kind of data formats and support such types of tools is the key to future enhancements of our approach. In addition, we also encountered support and usage limitations of our chosen modeling environment. AmmA is one of the most mature model engineering/DSL tool suites. However, it is not as mature as IDEs for general-purpose languages like Java or C#. Although it provides most of the basic modeling activities such as load, edit, build, store, and execution, lack of debugging makes it hard to identify errors in either the metamodel definition or the transformation specification.

**Acknowledgement**

We would like to thank the National Science Foundation (NSF-CAREER-0643725) for supporting this work.

**References**


WiELD-CAVE: Wireless ergonomic lightweight device for use in the CAVE

Joshua M. Hegie, Andrew S. Kimmel, Kelvin H. Parian, Sergiu M. Dascalu and Frederick C. Harris, Jr.
Department of Computer Science and Engineering, University of Nevada, Reno, 1664 N. Virginia St., Reno, NV, USA
Tel.: +1 775 784 6974; Fax: +1 775 784 1877; E-mail: {jhegie,akimmel,kelvinp,dascalus,Fred.Harris}@cse.unr.edu

Abstract. The goal of this project was to design a wireless input device for the CAVE virtual reality environment. The current solutions for this problem are not adequate, due to their high cost and wired nature. By eliminating these two problems, this team aimed to develop a more widely usable device that incorporates all of the functionality that other solutions have included, and more. For ease of use, the device is housed inside a pair of gloves and is capable of wirelessly communicating with the CAVE. The software driver that accompanies the device allows users to define a series of hand gestures, which are then mapped to either a button press in software – essentially, this allows users to manipulate the CAVE using only their hands and this device. Hopefully, the applications of this device will eventually expand beyond the researchers’ interests to the general public.

Keywords: CAVE, input device, wireless, glove

1. Introduction

Since the advent of computing technologies, being able to interact with an electronic environment has been the center focus of a plethora of research projects. Doing things such as clicking and dragging elements of the environment (i.e. through a mouse) or inputting elements into the environment (i.e. through the keyboard), was initially sufficient due to the simplicity of the environment. However, with the increasing popularity of virtual reality environments, such simplistic interactive input devices are no longer adequate. In order to take advantage of this more “advanced” environment, a more intuitive device is needed, and it is within this context that the project takes place in.

The goal of this project was to design a wireless ergonomic lightweight device (WiELD) that will be used to interact with the CAVE [1] virtual environment [2]. The basic functionality of the device is to allow users to wirelessly transmit “gestures” that will be recognized by a driver on the base station and subsequently translated into an “action” on the CAVE screens. The device is contained inside of a glove in order to provide users with a familiar interface for making the “gestures” (i.e. the gesture where the user contacts the forefinger with the thumb on the glove could translate into a “grab” action on the CAVE).

Modeling the WiELD-CAVE device according to the UML notation specified in [3] helped clarify the project goals, along with making the documentation of the project simpler and more unified.

The key components of this project include: a device with numerous inputs encapsulated in a glove, wirelessly communicating with a base station using an XBee [4] wireless chip, which acts as a wireless...
RS232 transmitter, the device is powered by a rechargeable lithium ion battery, all of the inputs utilize force sensitive resistors, and the corresponding driver for the device translates a set of contacts into the corresponding action in the CAVE.

Although there are several devices that are quite similar to the WiELD gloves, the current implementations of them are too expensive, wired to the base station, and/or have a limited number of recognizable gestures. By eliminating these problems, this team has developed a more easily accessible, as well as a more usable, device, which retains all the functionality of its “predecessors” along with additional functions.

The rest of this paper is structured as follows. Section 2 presents the requirements specification of the project. Section 3 presents the use case modeling. Section 4 presents architectural design. Section 5 presents the detailed design. Section 6 presents current status and future work. Section 7 presents our conclusion.

2. Requirements specification

Following standard software engineering guidelines [5,6], the main functional and non-functional requirements of WiELD-CAVE for both the hardware and software sides are presented below.

2.1. Functional requirements

The most important software and hardware functional requirements of WiELD-CAVE are:

1. The system shall output processed inputs in the form of VRPN [7] code.
2. The system shall provide users with the option of saving their configuration of the device.
3. The system shall provide users with the option of loading their previously saved configuration of the device.
4. The system shall provide users with the option of using a mouse to navigate through the system.
5. The system shall allow multiple sets of gloves to be used simultaneously.
6. The system shall allow for software synchronization of glove settings.
7. The system shall provide the status of the gloves to the user.
8. The system shall provide a setting for secure connection.
9. The system shall provide a setting for adjusting transmission power.
10. The device shall communicate wirelessly with the system.
11. The device shall be powered by rechargeable batteries.
12. The device shall allow each glove to operate individually.
13. The device shall have a display that indicates the device’s status.
14. The user shall be able to calibrate the device.
15. The device may gracefully power down upon a low battery status.
16. The device may have built-in motion tracking.
17. The device may have a built-in accelerometer.

2.2. Non-functional requirements

The most important software and hardware non-functional requirements of WiELD-CAVE are:

1. The system shall be implemented using C++. 

2. The system shall have a GUI implemented using the QT windowing toolkit.
3. The system shall run on the GNU/Linux operating system.
4. The device shall be programmed in C.
5. The device shall communicate using an Xbee wireless communications chipset.
6. The device shall be implemented using a Cortex microcontroller [9].
7. The device shall be encapsulated in a pair of ergonomic gloves.

3. Use case modeling

The functionality of WiELD-CAVE has been defined using use cases and scenarios as defined by the formal modeling process presented in [3] (Section 1 of this paper). The use case diagram, shown in Section 3.1, captures the entire functionality of WiELD-CAVE. This was done to help identify the mechanisms through which the user would interact with WiELD-CAVE. The use cases are compared to the requirements listed in Section 2 using the Requirements Traceability Matrix in Section 3.2, see Fig. 2.

3.1. Detailed use cases

Presented below are the Use Cases for WiELD-CAVE. A use case diagram, detailing how the use cases for both the hardware and the software interact, is presented in Fig. 1.

UC1. **sendData** – is the framework for collecting and transmitting data back to the base station.

UC2. **receiveData** – is responsible only for receiving data sent from the WiELD glove, acknowledging that the data has been received and passing the data on to the processing function.

UC3. **processData** – takes the user generated inputs and turns them into something the CAVE can understand. The user, via the send and receive functionality of the systems, initiates processing. The CAVE is the endpoint for this branch of the system.

UC4. **finishData** – is designed to ensure all input is received by the base station. Based on which gloves are connected to the base station. If one of the gloves has not sent its data, a request is sent to request that it provide the state of its contacts.

UC5. **pollDevice** – is a failsafe that can be called by the receiver if a device has failed to send data. It is the responsibility of this function to respond to finish requests and handle the entire cleanup associated with sending data.

UC6. **turnDeviceOn** – is invoked when the device is turned off and the power button is pressed. The device then has to power up the microcontroller. Once this is done the microcontroller will power up all of the peripherals and the Xbee wireless chip. Now the Xbee chip should pair with the base station. After all of this, the device will be ready to send data to the base station.

UC7. **turnDeviceOff** – is used to gracefully power down one of the WiELD gloves. The user initializes this by holding the power button down for a set amount of time. The wireless chip disconnects itself from the base station and the microcontroller is powered down.

UC8. **resetDevice** – is initiated by the user when the reset button on one of the WiELD gloves is pressed. This function is designed to power cycle the device, in the event that the hardware becomes unresponsive. This disconnects the wireless chip from the base station and powers down the microcontroller. Once everything is powered down, everything is returned to the ON state.
UC9. \textit{coordinate} – is invoked on the base station when one of the devices sends it data. The base station must read in 2 bytes from each device and then combine all of this data into a single numerical value. This is entered into the VRPN driver, so that programs running on the cave can access the state of the buttons.

UC10. \textit{chargeBattery} – is a function to allow the user to recharge the battery in the WiELD glove. This functionality is invoked by the user removing the device from the glove and placing it on the charging station. The device will then enter a low power state and begin charging. The microcontroller stays on to control the display, and once the battery has finished charging, the display turns off.

UC11. \textit{lowBattery} – is invoked to gracefully power the device down, so that the Xbee wireless chip disconnects from the base station, in order to prevent it from interfering with a new glove for that hand being introduced. The device has to watch the battery state once the battery is at or below a threshold. The user gets a warning once the battery is below a second threshold.
and the device powers down by disconnecting the Xbee wireless chip from the base station and powering down the microcontroller and all of the peripherals. Time acts on this because leaving the device on for any amount of time causes the charge in the battery to deplete.

UC12. *showValue* – is a functionality designed to show programmers what value they need to catch from the shared VRPN memory space. The user can generate input on one or more WiELD device and see which contacts are activated (have sufficient pressure applied) and the value that the program will place into the VRPN memory space.

UC13. *changeSetting* – is designed to allow a user at the base station to modify the firmware on the transmitters to change things such as the transmission strength or enable/disable encryption.

UC14. *modSetting* – is the glove interface to changeSetting. This must block the device from transmitting, update the data and then re-enable the device.

UC15. *calibrateDevice* – is used to let the user to calibrate how sensitive the inputs are. This will change the triggering threshold for the analog to digital converter, making it higher or lower, based on user preference.

3.2. Requirements traceability matrix

The Requirements Traceability Matrix (Fig. 2), detailed below (Requirements are in the left most column), shows how the use cases match up with the requirements listed in Section 2.

4. Architectural design

The layered architecture is one in which all data is passed through a series of hierarchical layers. A brief description of each subsystem utilized in WiELD-CAVE is as follows:
**C++ Libraries:** WiELD-CAVE’s driver is implemented using C++.

**C Libraries:** WiELD-CAVE’s firmware is implemented using C.

**Qt:** The driver GUI for WiELD-CAVE is handled through Qt windowing toolkit libraries and Qt framework.

**Help System:** A series of documents along with informative error messages which will assist in the user in trying to fix any problems encountered.

**Main Window:** The main window displays the status of the gloves in an easy to read format.

**LCD:** The LCD (Liquid Crystal Display) on the gloves shows the current device status and is located directly on the glove.

**Gloves:** Utilizes force sensitive resistors to generate signals that are passed to the transmitter.

**Transmitter:** The core of the hardware. This contains the analog-to-digital circuitry required to translate user input to digital logic, and interface with the wireless transmitter.

### 5. Detailed design

The class diagram for WiELD-CAVE, presented according to the specifications laid out in [3], is included in Fig. 3. This diagram lists all the classes in WiELD-CAVE as well as most of the major functions. All of the Qt variables and functions have been omitted in order to preserve room; however, this does not adversely affect the content of the diagram. The major classes for this system are WiELD Device, Driver and Maintenance.

**WiELD Device** is the class that encompasses the hardware element of the project, shown in Fig. 5. The primary responsibilities of this class are to notify the base station of changes in the button status. This is done through the private functions Interrupt1 and Interrupt2. These interrupt routines are triggered by the analog to digital converter, which is built into the ARM Cortex microcontroller, when it reads a voltage that has passed a threshold that is defined in code. Once this happens, the system checks the state of the transmit buffers to see if there are new buttons, using fast bitwise logic. In the event that there has been a change the send function is used to notify the driver of the change, otherwise nothing is sent to conserve power.

**The Driver class** is what is run on the VRPN server. This class is responsible for receiving and decoding the information that the WiELD device provides to it. The Receive function is automatically invoked by the USB XBee receiver. Once all data has been passed from the hardware to the driver, the Translation function is invoked. This function uses bitwise logic, bitwise AND and bitwise rotates, to determine which of the data sets it is currently inspecting. Once this information is available it is possible to place the received data into the correct storage variable. The BuildCall function is used to put all four of the storage variables into a single coherent value, so that it can be passed to VRPN. The nature of battery powered devices is, unfortunately, unstable. To prevent data loss there is a heartbeat function, QueryDevice, which is built into the driver. This makes it possible for the driver to correct for instances where the device loses connection, either from wireless interference or because the battery has run out. In the event that a heartbeat signal is not replied to the driver can zero out all of the information associated with the non-responsive device, effectively setting all buttons to the state where they are not pressed, and continue operating. Once the device comes back, it will be able to seamlessly begin communication with the base station.

**The Maintenance class** is distributed across both the hardware, and the software driver. This class allows the vital information, such as the signal strength or remaining charge of the battery, to be passed between the gloves and the driver. This allows for functionality that powers the gloves down
Fig. 3. Class diagram.
automatically, and with a clean disconnect from the driver, when the remaining charge on the battery drops below a defined threshold. This also allows for the driver, and the LCD display on the glove, to show how strong the signal strength for the wireless transmitters are, allowing the user to easily get out of a location that prevents the gloves from communicating with the base station.

6. Current status and future work

Currently, there is a working prototype of the WiELD-CAVE gloves. This prototype includes two gloves and two transmitters, the base station and all of the associated software. The software driver is able to receive and interpret the data, as well as relay it to a VRPN server.

The work done on this project can easily be expanded. Future work includes building motion tracking into the WiELD-CAVE gloves, using accelerometers and small gyros. Currently hand motions are being tracked using video tracking through IR cameras and reflective markers. While this does what it needs to, it is expensive, and can be cumbersome. The circuitry in the transmitter can be greatly minimized, allowing the main control unit can be built into the gloves instead of being enclosed in an armband and externally connected.

6.1. Hardware screenshots

Figure 4 shows the current implementation of the WiELD-CAVE gloves. On the fingertips are the force sensitive resistors. Figure 5 is a snapshot of the inner circuitry of the armband enclosure, behind
the Luminary ARM Cortex controller board is the circuitry that allows for the internal analog-to-digital conversion to work. Since the voltage that is permitted through the force sensitive resistors, even when the pressure is at the maximum allowed, is so small that it normally would not be detected. To compensate for this a series of amplifier circuits have been put behind the microcontroller. Not pictured is the interface from the glove itself to the enclosure, which is currently implemented as a ribbon cable [10] with a sufficient number of wires to send all of the data, for all nine of the inputs, to the controller.

6.2. Software driver screenshot

Figure 6 depicts the user interface [11,12] of the system and shows the connection status part of the driver, which updates in real time as the gloves’ status changes. The hands on the left and right hand sides of the interface allow the user to see which buttons are currently pressed. This can be useful if a button does not seem to be responding, the user can launch this application and test any button they like and see if it causes an update on the display. In the center is a graphical representation of the signal strength. This allows users to determine if there needs to be a settings change between low and high powered transmission. Along the bottom of the main section of the window is a bar that displays the synchronization status, which is the terminology used to describe the signal strength and a software
check of the coherence of the various settings on each glove, such as encryption. There is a second tab, which is not shown here, which allows the user to enable or disable the hardware encryption mechanism for the XBee transmitters. This functionality allows the user to encrypt the transmissions [13] between the gloves and the base station, making it easier to have multiple sets of gloves on the same wireless channel or to prevent other stations from reading the button presses that are being generated.

7. Conclusion

The WiELD-CAVE device which has been presented in this document is a unique and cost effective solution to intuitively interacting with the CAVE environment. WiELD-CAVE provides a high degree of flexibility in what can be taken as an input, as well as allowing developers to define what the device does with the inputs. The driver is light weight, requiring very little memory, and features a GUI that is simple to understand and intuitive to use.

There are nearly endless possibilities for expansion and improvements in the device. For example, with a little bit of modification, the WiELD-CAVE device could be used in gaming to interact with in-game objects. Other input architectures, such as Direct X, could be used as a backend for the device, allowing a much larger install base to use the WiELD gloves. System cross-compatibility should also be accounted due to the initial driver being limited to Linux distributions. These improvements will allow WiELD-CAVE to be used in more computing environments outside of virtual reality and the CAVE.

References

New strategies in the order picking process

Kemal Taljanovic\textsuperscript{a,\ast} and Adnan Salihbegovic\textsuperscript{b}
\textsuperscript{a}Archway Marketing Services, 7525 Cogswell Rd. Romulus, MI, USA
\textsuperscript{b}Faculty of Electrical Engineering, University of Sarajevo, Bosnia

Abstract. Increasingly important issues for every organization in the warehousing business are to achieve a high efficiency of all processes, minimize operational costs and maximize flexibility of the labor force. Among all processes, order picking is recognized to be the most significant and most costly activity and therefore tends to get the most attention. Key objectives in designing an order picking operation include increases in productivity and accuracy, and reduction of cycle time and replenishment costs. The aim of this article is to present a proposal of improved wave planning processes and modifications to the wave planning technique. With those modifications it is possible to reduce cost and time to process a wave by reducing the distance traveled by an order picker, the retrieval time per item and the replenishment costs.

Keywords: Forward pick area, order picking, wave planning, software agents

1. Introduction

Among all activities in a warehouse, order picking is considered to be one of the most significant. It accounts for roughly 55% of the total operating costs compared to traveling, which accounts for about 60% of the total picking costs. There are a variety of studies on methods, policies, principles and techniques developed to improve the overall order picking process. The decisions usually affect policies for the picking of the ordered materials, the routing of the pickers in the warehouse, and the storage schemes for the products in the warehouse. One common practice for increasing order picking performances is the separation of storage and picking activities. A separate picking zone, known as a forward pick area (FPA) or primary pick area, is a region of the warehouse in which materials to be delivered are concentrated within a small space, and where order picking is taking place. By reducing traveling time, the picking becomes more efficient. The trade-off is that the forward pick area must be replenished from a reserve zone in the warehouse.

The scope of this research is to address the reduction of the overall order picking costs by improving the wave planning process, thus reducing the cost of forward picking area replenishment. The algorithms developed have been coded, implemented, and tested for validation in the warehouse management system (WMS) which is discussed later in the article.

\ast Corresponding author. Tel.: +1 734 713 2264; E-mail: kemal\textsubscript{\ast}taljanovic@archway.com.

1472-7978/10/$27.50 © 2010 – IOS Press and the authors. All rights reserved
2. Review of order picking methods

Order picking can be defined as the activity by which a small number of goods are extracted from a warehousing system to satisfy a number of independent customer orders [2]. It is seen as the most labor-intensive and costly activity for almost every warehouse, where the cost of order picking is estimated to be as much as 55% of the total warehouse operating expense. As the order picking process involves significant cost and can affect customer satisfaction levels, researchers have proposed an increasing number of order picking process improvements [6].

Generally, all order picking methods are classified into “man-to-material” and “material-to-man” categories where each category has a number of different techniques.

2.1. Man-to-material picking method

‘Man-to-material’ is considered the basic solution for the order picking activity. During the picking activity, pickers retrieve from the picking locations, the items necessary to complete a single order or a batch of multiple orders (whether a single order or a batch picking policy was adopted) (Fig. 1). Picker travel time (in the form of walking, order picking vehicle or forklift truck) is very considerable for this picking method [5].

2.2. Material-to-man picking method

Travel time (Fig. 2) is the most significant part of the total time spent in order picking [5].

Total time to pick a group of orders is

\[ T_{\text{pick}} = \frac{D}{V} + nT_r \], where

- \( D \) – distance travel by order picker
- \( V \) – speed of the order picker
- \( n \) – number of items picked in a group
- \( T_r \) – retrieval time per item

Fig. 1. Man-to-material order picking method.
Order picking could be improved by reducing the travel time. Therefore, in many warehouses a separate forward picking area is set up to process orders, while a reserve storage area is used for replenishment i.e. material required to fulfill orders is moved to the forward picking area closer to the picker. The concept of the forward picking area is well known and described in subject matter related literature [7, 10]. In essence, it aims to make the pick area very small in order to reduce travel time. The number of locations available in the forward picking area is usually smaller than the total number of stock keeping units (SKU), thus forcing warehouse management systems (WMS) to resolve replenishment and slotting issues. Generally, the greater portion of time that a picker spends traveling during the ‘man-to-material’ order picking process, the higher the efficiency is gained in ‘material-to-man’ order picking process.

3. Forward picking area

The forward picking area increases the pick density by concentrating a large number of SKUs within a small physical place. The advantage of this method derives from the reduced picking costs (in terms of traveling time i.e. labor hours).

The basic issues in design of the forward picking area are [4,11]:

- Size, layout, and number of forward picking areas
- Which SKUs and how much of each SKU to store in the FPA
- SKU slotting within the FPA
- When and how much of each SKU to replenish

3.1. Cost of replenishment from reserved to forward picking area

The cost of replenishment from the reserve to the forward picking area is based mostly on the required number of replenishments, number of storage units being replenished, and timing (working shift) of restocking. If material Mi is stored in the FPA and the following applies
Fig. 3. Material-to-man order picking method.

\[ Q_i \] – quantity stored in the FPA
\[ YQ_i \] – total quantity picked per year
\[ v_i \] – volume stored in the FPA
\[ f_i \] – total volume picked per year

then number of replenishments per year is \( YQ_i/Q_i \) or \( f_i/v_i \).

Replenishment cost consists of the following costs [9]:
- Cost of traveling between FPA and reserve zone
- Cost of traveling within the reserve zones while searching for the material
- Cost of traveling within the FPA while replenishing
- Cost of the storage unit

3.2. Selection of SKUs to go into the forward pick area

The total cost of managing a SKU is the sum of picking costs and replenishing costs when picked from the FPA. Let the decision variable \( x_i \) be defined as

\[
x_i = \begin{cases} 
1 & \text{when SKU}_i \text{ picked from the FPA} \\
0 & \text{otherwise} 
\end{cases}
\]

\[ p_i \] – number of picks
\[ c_1 \] – picking cost from the FPA
\[ c_2 \] – picking cost from the reserve zone
\[ c_r \] – cost of replenishment
\[ f_i/v_i \] – number of replenishments per year

The objective function to minimize labor cost for managing a SKU in the FPA is [1,8]:

\[
\text{arg min} \sum_{i=1}^{n} (c_1 \cdot p_i + c_r \cdot (f_i/v_i))x_i + c_2 \cdot p_i (1 - x_i)
\]   (2)
and could be rewritten into an objective function that maximizes net savings when material is picked from the FPA:

$$\arg\min \sum_{i=1}^{n} ((c_2 - c_1) * p_i - c_r * (f_i/v_i))x_i$$ (3)

$$\arg\min \sum_{i=1}^{n} (s * p_i - c_r * (f_i/v_i))x_i$$ (4)

where $\sum_{i=1}^{n} v_i * x_i \leq 1$
$v_i \geq 0$
$x_i \in (0, 1)$
$s = c_2 - c_1$ represents saving when material is picked from the FPA.

The problem of selecting items that go into the forward pick area belongs to the class of multi dimensional knapsack or rucksack problems. A knapsack problem is a problem in combinatorial optimization where for any given set of items, each with a weight and a value, one has to determine the number of each items to include in a collection so that the total weight is less than a given limit, and the total value is as large as possible.

Various ranking lists help find materials with the largest impact on the processes inside a warehouse. Important stratifications lists are [1,2,10]:

- Ranking materials by the contribution to the total saving when picked from the FPA
- Materials are ranked by the coefficient of labor efficiency ($p_i/\sqrt{f_i}$), highest first, where $p_i$ is the expected number of picks of material $M_i$ in a given time period and $f_i$ is the total volume of material $M_i$ picked
- Ranking materials by the number of moved packages (case moving). This list helps planning resources for receiving, transfer, and replenishment because every package is individually handled
– Distribution of labor among materials, families of materials, warehouse zones, time (daily shift, days in a week, weeks in a year, etc)
– Seasonal variation in the type and quantity of the shipped materials
– Ranking by the numbers of trips to the picking locations
– Ranking by the quantity shipped.

3.3. Algorithm for initial selection of SKUs into the forward picking area

Let \( \text{cpo}_i \) (cube-per-order index, introduced by [3]) denotes an average shipped volume per order of material \( M_i \). Given that, the coefficient of labor efficiency \( (p_i/\sqrt{f_i}) \) becomes:

\[
\frac{p_i}{\sqrt{f_i}} = \frac{p_i}{\sqrt{p_i \cdot \text{cpo}_i}} = \sqrt{\frac{p_i}{\text{cpo}_i}} \approx \frac{p_i}{\text{cpo}_i}
\]

(5)

Conclusions drawn from this expression are:
– Materials that are frequently ordered (high \( p_i \)) in a low quantities (low \( \text{cpo}_i \)) are the best candidates to be moved to the FPA
– Slow moving materials should not be in the FPA due to the low savings (low \( p_i \))
– Large items are not placed into the FPA due to the high replenishment costs i.e. low savings of being picked from the FPA (derived from the expression for net saving when a SKU is picked from the FPA).

An algorithm for the initial material selection into FPA and number of locations allocated to each material is:

Step 1. Create a set \( M \) whose elements are tuples (SKU, coefficient of labor efficiency, requested volume)
\[ M = \{(m_i, k_i, v_i)\}, i = 1, \ldots, N \]

Step 2. For each element \( m_i \) calculate the coefficient of labor efficiency \( k_i = (p_i/\sqrt{f_i}) \)

Step 3. For each element \( m_i \) calculate the requested volume \( v_i \)

Step 4. Order set \( M \) by descending value of element \( v_i \)

Step 5. Create a subset \( S \in M \) of elements from set \( M \) which will provide the maximal savings when moved to the FPA. Selection of elements of \( S \) from the set \( M \) is an iterative process which calculates maximal net savings when first element \( m_i \in M \) is moved to the FPA, when the top two elements \( m_i \in M \) are moved to FPA, etc. The process is repeated until such subset \( S \in M \) is found that net savings are maximal

Step 6. Remove all elements \( m_i \in S \) having required volume \( v_i \) less than minimal volume \( c_r \cdot f_i / s \) * \( p_i \) to be placed in the FPA

Step 7. For all remaining elements \( m_i \in S \) calculate a number of required locations dividing required volume by physical volume of the location in the FPA.

3.4. A cost-based model of warehouse order picking process

The model presented below reflects several typical costs involved in the order picking process.
Indexes used: materials are denoted with \( i = 1, \ldots, I \)
locations in the forward picking area are denoted with \( j = 1, \ldots, J \)
Variables:

\[ x_{i,j} = \begin{cases} 
1 & \text{if material } m_i \text{ received into reserve zone, replenished and picked from the FPA where occupies at least } j \text{ locations} \\
0 & \text{otherwise} 
\end{cases} \]

\[ y_i = \begin{cases} 
1 & \text{if material } m_i \text{ received into the FPA and picked from the FPA only} \\
0 & \text{otherwise} 
\end{cases} \]

\[ z_i = \begin{cases} 
1 & \text{if material } m_i \text{ received into the reserve zone and picked from the reserve zone only} \\
0 & \text{otherwise} 
\end{cases} \]
\[ u_{i,j} = \begin{cases} 
1 & \text{if material } m_i \text{ received into the reserve zone, replenished and picked from} \\
& \text{the working area where occupies at least } j \text{ locations} \\
0 & \text{otherwise} 
\end{cases} \]

Constraints used: \( x_{ij}, y_{i}, z_{i}, u_{i} \geq 0 \), for each \( i, j \)

\[ \sum_{i} (x_{i1} + y_{i} + z_{i} + u_{i}) = 1 \]

\[
\begin{align*}
CR_{w} & < CR_{a} < CR_{f} \\
PC_{w} & < PC_{f} < PC_{r}
\end{align*}
\]

Objective function is

\[
\min \sum_{i} CR_{a} \cdot x_{i1} + \sum_{i} \sum_{j} CR_{f} \cdot x_{ij} \cdot q_{ij} + \sum_{i} CR_{w} \cdot u_{i} \cdot q_{i} + \sum_{i} PC_{f} \cdot E(p_{i}) \cdot (x_{i1} + y_{i})
\]

\[ + \sum_{i} PC_{r} \cdot E(p_{i}) \cdot z_{i} + \sum_{i} PC_{w} \cdot E(p_{i}) \cdot u_{i} \]

\[ (6) \]

4. A new strategies in the order picking process

Wave planning is a quality methodology developed for analyzing, grouping and releasing of outbound orders together in order to achieve more efficient picking and more balanced flow throughout the warehouse. A pick wave is a set of orders selected for picking/shipping during a certain time window where that time window is a portion of the shift (e.g., eight 1-hour time windows in one shift) during which a pick wave is released and fully processed. The objective of wave planning is to obtain a continuous throughput across replenishment, picking, and shipment activities. A wave creating process in the commercially available WMS systems consists of the following workflow steps:

- Wave selection – select orders applying selection criteria (area of the warehouse, similar attributes of orders, similar attributes of material, destination, carrier, appointment, etc.)
- Wave replenishment – move all the material required for the wave to the picking area
- Wave release – create the picking tasks and assign them to the pickers.

Several modifications to the wave planning technique could significantly increase the order picking efficiency. With those modifications it is possible to reduce time to process a wave by reducing distance traveled by order picker and retrieval time per item and, at the same time, increase the number of items picked in a wave. In the essence of the proposed modifications is an attempt to preserve quantity in the forward picking area by recognizing groups of orders that could be picked efficiently outside of the FPA. It will reduce number of replenishment trips i.e. reduce the cost of the replenishment.

Proposed modifications of the wave planning technique are:

- Materials individually packed in big containers and ready to be shipped (so called ship alone items) should not be kept and replenished into the forward picking area. When ordered, such materials need to be directly transferred from the reserve area to the shipping area. Therefore, it is necessary
to modify picking documents into shipping labels (for the ship alone items) and pick lists (for the remaining items). Keeping ship alone items outside of the FPA saves a valuable space inside the FPA for the other items and eliminates cost and time for replenishing them into the FPA.

– Contrary to many other solutions where an order picking process starts upon replenishment of all required material, the proposed solution carefully orchestrates pick release and replenishment tasks in order to eliminate a resources’ idle time (waiting for replenishment to be completed) and speeds up throughput through the warehouse. It is not necessary to wait for all required material to be replenished in order to start the order picking process. Either the WMS itself or users should further analyze the wave (selected orders) to determine what groups of orders could be fulfilled immediately and what groups of orders require materials to be replenished into the FPA.

– Contrary to many other solutions where all orders tend to be picked from the forward picking area, the proposed solution is able to pick and pack orders inside working zones (production line) where materials have been moved. If a selected wave has a number of identical orders above the certain threshold, then a more efficient process would be to move required material directly from the reserve area to the production line and fulfill orders there. This modification eliminates replenishment and search times. At the same time, material leftovers could be moved to the forward picking area instead of moving back to the reserve area.

– Proposed solution supports more than one forward picking areas. Secondary forward picking areas could be specialized for the very large truck orders. If fulfilled from the primary forward picking area, such order could easily deplete many materials to zero quantity and cause numerous replenishment trips, thus increasing a total cost of the order picking process.

– Proposed solution supports containerization. Contrary to many other solutions where all materials for an order are first picked and then, if necessary, separated into many packages, the proposed solution creates as many picking documents as packages. The process takes into consideration weight and cube capacities of the packaging materials. This change reduces an order cycle time by allowing many pickers to concurrently fulfill the same order. Also, time and resources for packaging are no longer needed.

5. Benefits of the proposed WMS systems design

From the warehouse operations perspective, the new, improved operational strategy of the WMS, envisioned in this research paper, helps improve efficiency of the warehouse operations by reducing travel time and number of replenishment trips, and by increasing utilization and efficiency of the resources. Instead of having start and cut off times for the warehouse processes, the new WMS continuously observes events in the warehouse and takes appropriate actions. Contrary to many others solutions where one big batch of orders is created, the new WMS creates many small batches of orders. It does not wait for all materials to be replenished; the system assigns fulfillable batches to the pickers as soon as they request work and carefully synchronizes the timing of the batch release and the replenishment. A new WMS design offers the following improvements:

– picking and replenishment processes are carefully orchestrated
– when needed, material is dynamically moved to the forward picking area
– material is dynamically moved to the production assembly line when the number of identical orders exceeds a given threshold
– material is dynamically moved to the secondary forward picking area for every large order
– preserve material in the FPA
– picking containerization is supported.
6. Post implementation results

For an existing WMS system a new wave planning and execution process was developed and implemented (Figs 6, 7, 8). Effects of the change were observed in four warehouses. Optimization rules considered in this process were:

- Minimization of replenishment cost
- Minimization of the labor costs
After the proposed changes, as described in the paragraph 4, were implemented, the following improvements have been achieved:

- Maximization of resource utilization

Wave planning has a parameter rich user interface form. It allows users to select either all or filter out desired groups or orders for further processing (Fig. 7). If the button ‘Query by Order Groups’ was pressed then selected orders are clustered into groups by number of line items. (Fig. 8). For each group of orders, the system determines whether a replenishment is required or not. Users are able to create picking documents for some groups and replenishment transfer tasks for others (when required). Picking tasks can be grouped together by assigning the same wave ID. If the button ‘Query by SKU Groups’ was pressed then selected orders are clustered into groups by combinations of SKUs (Fig. 9). For each group availability of SKUs in the primary picking zone or selected picking zone is indicated. Users have ability to process a group of orders outside of the primary picking zone by selecting a working area. In such cases, the system creates a task to transfer material from the storage zone to the working zone, and picking documents that point to the working zone.
7. Conclusion

In this paper we presented a new strategy for an improved wave planning process and the order picking process. The main objective of this research was to verify the expected effects (the reduction of the overall picking time) of an improved wave planning process in the real working environment of four warehouses. As the results shown in Table 1 indicated, after the proposed modifications were implemented in existing WMS system, a significant reduction in the head counts and idle times were achieved. By eliminating idle time, productivity of each worker was increased 6.25%, and labor cost was reduced about 60% due to fewer workers assigned to the tasks.

Our future work will focus towards design and development of an agent-based WMS system. It can effectively combine all the advantages of distributed systems such as scalability, reliability, and reusability with artificial intelligence (AI) techniques. Specifically, in logistics, software agents can be applied to enhance existing WMS with the aim of superseding the intelligence of the human labor force, integrating automated equipment with warehouse processes, and creating unattended autonomous processes to continuously observe changes within the warehouse and take actions towards a given goal. The presented algorithms for improved wave planning strategy are examples of the application of an artificial intelligence in the decision making processes implemented within modern WMS systems.

References

Integrating formal methods with domain analysis

Laura Felice\textsuperscript{a,*}, Marfa Carmen Leonardi\textsuperscript{a}, Marfa Virginia Mauco\textsuperscript{a}, Germ\`an Montejano\textsuperscript{b}, Daniel Riesco\textsuperscript{b} and Narayan Debnath\textsuperscript{c}

\textsuperscript{a}Universidad Nacional del Centro, Campus Universitario, 7000 Tandil, Argentina
\textsuperscript{b}Universidad Nacional de San Luis, Ejército de los Andes 950, 5700 San Luis, Argentina
\textsuperscript{c}Winona State University, Winona, MN, USA

Abstract. The use of formal methods should help to achieve a high degree of confidence that a system will conform to its specification, enhancing in consequence software quality and reliability. However, a general acceptance of formal methods among software engineers is still some way off because formal methods are usually only accessible to specialists and they do not have developed in depth strategies for the first stages of development. This is also valid to Domain Analysis, because its first stage is to capture the knowledge of a particular domain, making necessary to have a model comprehensible by software engineers and domain experts. In order to address this problem and take advantage of formal methods, we suggest integrating the phase reusable Domain Analysis into the RAISE Method, combining Domain Analysis notions with a formal language in the early steps of software development process. In this paper, we present a set of heuristics to fruitfully use knowledge represented in a Domain Analysis model to derive a formal specification in the RAISE Specification Language.

Keywords: Domain analysis, feature model, RAISE method, RAISE specification language

1. Introduction

Nowadays we need software in many aspects of our life. As formal methods offer a wide spectrum of possible paths towards designing high-quality software, they are receiving increasing attention in the academia and the industry, especially where safety or security is important [25]. By using formal methods early in the software development process, ambiguities, incompleteness, inconsistencies, errors, or misunderstandings can be detected, avoiding their discovery during costly testing and debugging phases. However, one of the problems with working with formal specifications is that they are hard to give a first specification comprehensible to stakeholders, and even to non-formal specification specialists. This is particularly inconvenient during the first stages, when interaction with stakeholders is very important. As our work is focused on Domain Analysis (DA) [15], we consider domain analysis modeling as the first stage in the software development process.

The information gathered during DA is heavily based on informal and semi-formal models as stakeholders, one of the main sources of this information, must be able to read and understand the results of these phases [1,18]. Therefore, a good formal approach should use both informal and formal techniques [2, 26].

*Corresponding author. E-mail: \{lfelice,cleonard,vmauco\}@exa.unicen.edu.ar.
A well-known formal method is the RAISE Method, which has been used on real developments [6,20]. RAISE is an acronym for “Rigorous Approach to Industrial Software Engineering” [10]. The RAISE Method includes a large number of techniques and strategies for doing formal development and proofs, as well as a formal specification language, the RAISE Specification Language (RSL) [9], and a set of tools [11]. However, it does not include any strategy for the first stages of development that addresses the problems mentioned before.

Feature models have been widely adopted to capture, organize and reuse the requirements of a set of similar applications in a software domain [27]. The DA models chosen are the Feature Model initially described by Kyo Kang in FODA (Feature Oriented Domain Analysis) [15], and it has been applied in a range of business and technical domains. In this work, we use cardinality-based feature modeling [4, 5], which extends the original feature modeling from FODA with feature and group cardinalities, feature attributes and feature diagram references.

In the early stages of software family development, feature models provide the basis for scoping a system family by recording and assessing information. We consider the introduction of feature model for DA is a crucial task because it enhances the possibility of reusing specifications in the future.

In order to integrate DA modeling into RAISE, in this paper we propose a set of heuristics to derive an initial RSL specification from a feature model. This proposal is a transformation that, guided by the feature selection and the groupings among features, derives an initial specification in RSL. This work presents an extension of [7].

The remainder of the paper is organized as follows. Section 2 gives an overview of the RAISE Method; the concepts of feature oriented DA are presented in Section 3. The core of the paper is in Section 4, where we describe the transformation strategy proposed. In Section 5 we exemplify the strategy. Finally, in Section 6 we present some conclusions and outline possible future work.

2. The RAISE method

RAISE gives its name to a wide spectrum specification and design language, the RAISE Specification Language (RSL), an associated method, and an available set of tools to help writing, checking, printing, storing, transforming, and reasoning about specifications. Complete descriptions of RSL and the RAISE Method can be found in the corresponding books [9,10], while the tools are described in [11] and they can be downloaded from UNU-IIST’s web site (www.iist.unu.edu).

There are two main activities in the RAISE method: writing an initial specification, and developing it towards something that can be implemented in a programming language [12]. Usually the first RSL specification is an abstract, applicative and sequential one, which is later developed into a concrete specification, initially still applicative and then, imperative, and sometimes concurrent. The method provides many guidelines to hierarchically structure a specification, aiming at encouraging separate development and step-wise development. A specification in RSL is a collection of modules. A module is basically a named collection of declarations; it can be a scheme or an object. Each module should have only one type of interest, defining the appropriate functions to create, modify, and observe values of the type. However, the kernel module concept is that of a class expression. A basic class expression is a collection of declarations enclosed by the keywords class and end and it represents a class of models. Objects and schemes are defined using class expressions. The following definitions exemplify some of the concepts defined above.
context: STOREFRONT, BACKOFFICE

scheme E-SHOP =

class

object
  SF: STOREFRONT, /*mandatory */
  BO: BACKOFFICE, /*mandatory */

type
  E-shop::
    storefront: SF.StoreFront
    backoffice: BO.BackOffice
end

A typical applicative class expression contains type, value, and some axiom definitions. Axioms may be used to constrain the values or to model invariants.

RSL is a typed language. This means each occurrence of an identifier representing a value, variable, or channel must be associated with a unique type. Besides, it must be possible to check whether each occurrence of an identifier is consistent with a collection of typing rules.

A type is a collection of logically related values, and it may be specified by an abstract or a concrete definition. An abstract type, also referred to as a sort, has only a name. It is a type we need but whose definition we have not decided yet. A concrete type can be defined as being equal to some other type or using a type expression formed from other types.

Values are constants and functions. Their definition must include at least the signature that is a name, and types for the result, and for the arguments, in case of a function. A function is a mapping from values of one type to values of another type, and it can be total or partial. It is total when it is defined for every value of the arguments, and it is considered partial when it is not known to be total.

3. Feature-Oriented Domain Analysis

A domain model is the result of the analysis of commonalities and variabilities of systems within a domain. It is a high level description of the application family, providing a framework for describing the essential characteristics. Examples of more relevant DA methods include Feature-Oriented Domain Analysis (FODA) [13], Organization Domain Modeling (ODM) [24], FeatureRSEB [13], Feature-Oriented Reuse Method (FORM) [16]. They support the notion of feature-oriented. This is a concept based on the emphasis this method places on finding the features or functionalities usually expected in applications for a given domain. Feature models are a modeling notation to represent the variability in a system family and describe all valid configurations.

The model constructed captures commonality as an AND/OR graph. Then, this model is used to define parameterized reference architectures and appropriate reusable components which are instantiated during actual application development. Feature models are able to describe the aspects, commonalities or characteristics of a system and include variability modeling for system families. Commonalities can be modeled by common features (mandatory features whose ancestors are also mandatory), and variabilities can be modeled by variant features, such as optional, alternative, and or-features.

A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [15]. A feature is detailed in any number of other features (subfeatures). Mandatory features describe detailed aspects that the parent feature must support, while optional features may be selected
when creating a concrete system from a feature model. Alternative features have a multiplicity similar to the UML multiplicity, defining how many of the features must or may be selected [18]. In addition to the hierarchical relationship, a constraint relation defines if a feature requires, conflicts, modifies or excludes with any other features. Figure 1, extracted from [17], summarizes the notation.

The feature requested by a stakeholder is called a concept feature, it is a root node of the feature diagram and all features are represented as child nodes. The hierarchical relationships mandatory, optional, and alternative are represented by different edges between the nodes. A simple line with a filled circle represents a mandatory relationship, while the optional is represented with a line ending with an empty circle. Arcs spanning two or more edges of the feature nodes depict a set of alternative features. The arc is annotated with the multiplicity of the alternative. Normally, a grouped feature has [0..1] as its default cardinality. The cardinalities [1..1] (xor group), [1..k] (or group) and [0..0] are used for features that were selected or eliminated, respectively, from a group during specialization.

Figure 2 illustrates a fragment of [21, Fig 4.1] which represents the simplified Store Front of an e-commerce domain analysis. Store Front is the interface that the customer uses to access to e-shop. The features here are related to the interface, visible to the customer and consist of a set of functional features representing the home page, catalog, and buy path and they are modeled as mandatory features. By the other hand, the Store Front may also include registration, wish list, customer service and/or user behaviour tracking capabilities and they are modeled as optional features. Bellow HomePage feature are their subfeatures. The content of a home page can be generated statically or dynamically. A page can contain both static and dynamic content, meaning that the feature group is inclusive-or. Dynamic content is for content which changes frequently. It is generated on demand and every customer receives a customized home page for each session. There are two required parameters: the content type and the variation source that are modeled as part features mandatory. Two common content types are the welcome message and special offers. Content type feature can contain both, so that the feature group is inclusive-or.

Furthermore, the Business management shown in Fig. 3, deals with aspects related to the e-shop’s operation. Business management requires the administration features, which is modeled as mandato-
Inventory tracking and Targeting are considered optional features which can augment the business management capabilities.

Targeting refers to marketing efforts which focus on meeting the needs and preferences of individual users. Targeting is defined by: Targeting criteria, Targeting mechanisms and Display and notifications. These are modeled as mandatory features of the feature group.

Targeting mechanisms feature provides methods for implementing a marketing effort. The mechanisms can be split into categories: advertisements, discounts and sell strategies. These features are modeled as inclusive-or feature group.

In addition to the parental relations between features, a feature model can also contain cross-tree constraints between couples of features. They define relationships between features which are not expressed by the tree of the model and they take the form of an implication. A strong constraint refers to
Fig. 4. Requires relation.

A requires B means that the selection of A implies that B must also be selected. For example: discount feature provides mechanisms for defining and managing price adjustments on products. The context says that special offers consist of a series of discounts (Fig. 4).

4. Integrating domain analysis into RAISE Method

RAISE Method is not focused on DA [8]. Thus, our work is centered on the incorporation of DA to the RAISE Method in order to specify a family of systems to produce qualitative and reliable applications in a domain, promoting early reuse and reducing development costs. We propose to use a feature model to represent the domain analysis because they facilitate the customization of software requirements. In DA, features and relationships between features (called domain feature model) are used to organize the requirements of a set of similar applications in a software domain [3].

The strategy we propose starts by defining this feature model following one of the several proposals that facilitate feature models’ construction [13,15,27] This model will be then transformed into a RSL specification that can be later developed into a more concrete one to automatically obtain a prototype to validate the specification by using the RAISE Tools [11]. In the following section we describe the strategy in more detail.

4.1. From feature-oriented models to RSL specifications

In this section we discuss how Feature Modeling serves as a guideline to identify RSL reusable types. We use the structural information of the feature models to derive RSL constructs based on [5] where the features are typed. Mandatory features, Optional features, OR features and alternative features must be differentiated. We propose a strategy which consists on a set of heuristics to derive an initial hierarchy of RSL types from a feature model. The goal of each heuristic is to identify an element or a combination of connected elements of the feature model and give the equivalences in terms of RSL constructs.

To follow one of the principles proposed in the RAISE Method [10], the heuristics first identify types, obtaining a set of RSL abstract types, that later are developed into more concrete ones. The heuristics to develop types analyze feature model in order to complete the specification of each type, defining for example attributes, cardinalities, and axioms. This way of defining types follows one of the key notions of the RAISE Method: the step-wise development. The replacement of an abstract type by a more concrete one follows the implementation relation. Implementation is very important because if an initial
specification meets the requirements and all its developments follow the implementation relation, then they all meet the requirements.

We can summarize the transformation from feature model into RSL specifications in the following steps:

- Transform the System feature model into a hierarchy of modules.
- Transform each Concept into a RSL scheme with a type of interest.
- Transform each feature into a RSL scheme (generally).
- Analyze each kind of feature: Solitary, PartFeature, AggregateFeature, GroupedFeature and FeatureGroup in order to complete the RSL specification. For example if the feature is SolitaryFeature, we have to associate with the same multiplicity (featureCardinality) to the parent feature.
- Transform a mandatory typed feature (primitive type) as a leaf of a tree into a RSL type in the class that represents the parent feature.
- Transform each FeatureGroup into the parent scheme of a set of features of GroupedFeature using the FeatureGroup name.
  * Transform each GroupedFeature into a scheme expressing the cardinality (lower and upper value).
- Transform each AggregateFeature into the parent scheme of a set of features of PartFeature using the AggregateFeature name.
  * Transform each PartFeature into a scheme expressing the cardinality (lower and upper value).

The transformation described before is achieved through a set of heuristics that guide the mapping between the Feature Model and RSL constructs. Heuristics are presented in two tables. Table 1 presents heuristics (named with the prefix MFR) relating features -F- with RSL constructs. Table 2 shows the heuristics to complete RSL type definitions based on relationships among features (the natural way to do these mappings is by means of axioms); we use the prefix MRR to mean mapping relations -R- between features to RSL specifications.

### 5. Applying the transformation to a case study

We exemplify the application of some heuristics by considering the feature model in Fig. 2. The following is a first, and still incomplete, definition of these RSL modules.

```plaintext
MFR9 AggregateFeature StoreFront -
STOREFRONT scheme:
Each part is previously defined as modules:

- HomePage part name (mandatory)
- UserBehTrack part name (optional)
- Catalog part name (mandatory)
- WishList part name (optional)
- BuyPath part name (mandatory)
- CustomerService part name (optional)
- Registration part name (optional)

context: ...
scheme STOREFRONT
class object
  HomePage, /*mandatory*/
  UserBehTrack, /*optional*/
  Catalog, /*mandatory*/
  WishList, /*optional*/
  BuyPath, /*mandatory*/
  CustomerService, /*optional */
  Registration, /*optional */
...,
end
```
Table 1  
Mapping from features to RSL specification concepts

<table>
<thead>
<tr>
<th>MFRid</th>
<th>Elements of a Feature diagram (FD)</th>
<th>Correspondence to RSL language</th>
</tr>
</thead>
</table>
| **MFR1** | FD=concept + {feature} (module) hierarchy | \*RSL scheme, a class expression with a type of interest /*
| **MFR2** | Concept | scheme CONCEPT= class 
| **MFR3** | Feature= {Feature Group | GroupedFeature? | AggregateFeature | PartFeature} (SolitaryFeature) | scheme FEATURENAME= type 
| **MFR4** | GroupXFeature (xor group) | /* Root feature of a xor-group (cardinality<0-1>) */
| **MFR5** | GroupOFeature (or group) | /* Root feature of an or-group (cardinality<1-k>) */
| **MFR6** | GroupORFeature (or group <i-j>) | /* Root feature of an or-group (cardinality<i-j>) */
| **MFR7** | GroupedFeature | /* extend is a RSL operator indicating the feature belongs to a group*/

---

| MFR10 | PartFeature (mandatory) | /* Parts of an aggregate with cardinality [1..1] */
| **MFR11** | PartFeature (optional) | /* Parts of an aggregate with cardinality [0..1] */

---
### Table 1, continued

<table>
<thead>
<tr>
<th>MFRid</th>
<th>Elements of a Feature diagram (FD)</th>
<th>Correspondence to RSL language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFR12</td>
<td>SolitaryFeature <strong>mandatory</strong></td>
<td>class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cardinality:: lower: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>upper: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SolitaryFeature::</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name: SolitaryFeature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cardinality: Cardinality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>MFR13</td>
<td>SolitaryFeature <strong>optional</strong></td>
<td>class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cardinality:: lower: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>upper: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SolitaryFeature::</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name: SolitaryFeature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cardinality: Cardinality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>MFR14</td>
<td>SolitaryFeature <strong>mandatory clonable</strong></td>
<td>class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cardinality:: lower: n</td>
</tr>
<tr>
<td></td>
<td></td>
<td>upper: m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SolitaryFeature::</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name: SolitaryFeature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cardinality: Cardinality</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>MRRd</th>
<th>Relations in a Feature Diagram</th>
<th>Correspondence to RSL language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR1</td>
<td>Requires relation</td>
<td>type Relation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rel={ r:Relation-well_formed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(r)?}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>requires: FeaName x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FeaName → Bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>requires(requester,supplier)≡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is(selected)(requester) ⇒</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is(selected)(supplier)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>MRR2</td>
<td>Conflicts relation</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>conflict: FeaName x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FeaName → Bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>conflict(requester,supplier)≡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is(selected)(requester) ⇒</td>
</tr>
<tr>
<td></td>
<td></td>
<td>~is(selected)(supplier)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>MRR3</td>
<td>Modifies relation</td>
<td>type</td>
</tr>
</tbody>
</table>

### MRRd | Relations in a Feature Diagram | Correspondence to RSL language |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR4</td>
<td>Excludes relation</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axiom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exclude: FeaName x FeaName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→ Bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exclude(source,target)≡</td>
</tr>
<tr>
<td></td>
<td></td>
<td>~is(selected)(target) ⇒</td>
</tr>
<tr>
<td></td>
<td></td>
<td>~is(selected)(source)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. . .</td>
</tr>
</tbody>
</table>

### MRRd | Relations in a Feature Diagram | Correspondence to RSL language |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MFR6 GroupORFeature HomePage –
HOMEPAGE scheme:
HomePage is defined with two cardinality types.
PartCardinality (mandatory part of an Aggregate) and
GroupCardinality (root feature of an or-group with
cardinality i-j)

\[
\begin{align*}
\text{context: ...} \\
\text{scheme HOMEPAGE} \\
\text{class} \\
\text{type} \\
\text{PartCardinality:: lower: 1} \\
\text{upper: 1} \\
\text{GroupCardinality:: lower: i} \\
\text{upper: j} \\
\text{HomePage::} \\
\text{partcardinality : PartCardinality} \\
\text{groupcardinality:} \\
\text{GroupCardinality} \\
\text{...} \\
\text{...} \\
\end{align*}
\]

MFR9 AggregateFeature Registration –
REGISTRATION scheme:
Registration is defined as a part of StoreFront aggregate
and
as an aggregatefeature with the following modules:
– RegistrationEnforc part name (mandatory)
– RegistrationInform part name (mandatory)
– UserBehTracInfo part name (optional)

\[
\begin{align*}
\text{context: ...} \\
\text{scheme REGISTRATION} \\
\text{class} \\
\text{object} \\
\text{RegistrationEnforc, /*mandatory*/} \\
\text{RegostInform, /*mandatory*/} \\
\text{UserBehTrackInf, /*optional */} \\
\text{type} \\
\text{PartCardinality:: lower: 0} \\
\text{upper: 1} \\
\text{Registration::} \\
\text{partcardinality : PartCardinality} \\
\text{...} \\
\text{...} \\
\end{align*}
\]

MFR11 PartFeature (optional) UserBehTrack –
USERBEHTRACK scheme:
UserBehTrack is defined as a part of an aggregate with
cardinality [0..1]

\[
\begin{align*}
\text{context: ...} \\
\text{scheme USERBEHTRACK} \\
\text{class} \\
\text{type} \\
\text{PartCardinality:: lower: 0} \\
\text{upper: 1} \\
\text{UserBehTrack::} \\
\text{partcardinality : PartCardinality} \\
\text{...} \\
\text{...} \\
\end{align*}
\]

MFR10 PartFeature (mandatory) Catalog –
CATALOG scheme:
Catalog is defined as a part of an aggregate
with cardinality [1..1]

\[
\begin{align*}
\text{context: ...} \\
\text{scheme CATALOG} \\
\text{class} \\
\text{type} \\
\text{PartCardinality:: lower: 1} \\
\text{upper: 1} \\
\text{Catalog::} \\
\text{partcardinality : PartCardinality} \\
\text{...} \\
\text{...} \\
\end{align*}
\]
MFR11 PartFeature (optional) WishList –  
WISHLIST scheme:  
WishList is defined as a part of an aggregate  
with cardinality [0..1]

```
context: ...
scheme WISHLIST
class
type
  PartCardinality:: lower: 0
  upper: 1
WishList::
  partcardinality: PartCardinality
...
end
```

MFR10 PartFeature (mandatory) BuyPath –  
BUYPATH scheme:  
BuyPath is defined as a part of an aggregate  
with cardinality [1..1]

```
context: ...
scheme BUYPATH
class
type
  PartCardinality:: lower: 1
  upper: 1
BuyPath::
  partcardinality: PartCardinality
...
end
```

MFR11 PartFeature (optional) CustomerService –  
CUSTOMERSERVICE scheme:  
CustomerService is defined as a part of an aggregate  
with cardinality [0..1]

```
context: ...
scheme CUSTOMERSERVICE
class
type
  PartCardinality:: lower: 0
  upper: 1
CustomerService::
  partcardinality: PartCardinality
...
end
```

We apply MRR1 heuristic to “requires” relation that links SpecialOffers and Discounts features. This  
relation describes that the binding of one variant implies the need of another variant (required variant).  
Therefore, we specify this constrain as follows:

MRR1 Requires relation SpecialOffers –  
DISCOUNTS scheme:  
Being feature A: SpecialOffers and feature B: Discounts,  
thus A requires B.

```
context: DISCOUNTS
scheme SPECIALOFFERS
class
type
  Net::
    cardinality: Cardinality
...
axiom
  requires (SpecialOffers, Discounts):
    is_selected(SpecialOffers) \Rightarrow
    is_selected(Discounts)
...
end
```
The RSL specification obtained by applying the heuristics is incomplete. Therefore, the participation of a software engineer is unavoidable in order to complete it.

6. Conclusions

Computers are used for tasks where failure has severe consequences. Therefore, reliability is essential for both software and hardware. An important technique to aid in increasing software reliability is the use of formal methods [10]. Formal methods help to avoid analysis ambiguities providing in consequence a correct software development process based on mathematical proofs. However, the models obtained by these methods are unfamiliar for business people involved in the first stages of the software development process, when interaction with stakeholders is crucial to understand and communicate the problem. This is also valid for DA. In order to contribute to incorporate DA into RAISE Method, we have presented a proposal to be used in the context of RAISE development using feature modeling. The use of feature model is motivated by the fact that stakeholders often speak about product characteristics in terms of “features the product has and/or delivers”, using them to communicate their ideas, needs, and problems. Feature Model structures domain components in a way that can be used as the basis for a RSL specification derivation. In this paper we have presented an initial specification of a system domain representing a mapping from real-world concepts onto RSL specifications. RSL specifications derived are in an applicative sequential style. In addition, it is an important issue that the relationships between features have an exact correspondence to RSL schemes. Feature models represent the commonality and variability in a system. Structural and Behavioral (services and operations) features can be differentiated. In this work, we have only considered the structural mappings. RSL specifications derived do not have function definitions. For example, FeatureRSEB method [13] proposes a division into system components and the generation of a high level use case model. This method extends the use-case modelling of RSEB [14]. We plan to enhance our proposal by refining and completing the heuristics in order to find the behavioral mappings, and then derive RSL functions. Besides, we have to consider the definition of a set of heuristics to organize all the RSL types derived in a hierarchy of modules.

References


