The Computer Journal Advance Access published November 12, 2014

© The British Computer Society 2014. All rights reserved.
For Permissions, please email: journals.permissions@oup.com
doi:10.1093/comjnl/bxu126

A Methodology for Resolution Mapping for Cross-Resolution Simulation using Event-B

Ahmet Kara[1,2]*, Halit Oğuztüzün[1] and M. Nedim Alpdemir[2]

[1]*Department of Computer Engineering, Middle East Technical University, Ankara, Turkey*
[2]*Tübitak Bilgem İltaren, Ankara, Turkey*
**Corresponding author: ahmet.kara@tubitak.gov.tr*

This paper proposes a software engineering solution for implementing simulations via the composition of models at different resolution levels with the help of formal methods. Our solution provides a systematic methodology that offers a well-defined sequence of stages to obtain executable converters for entity resolution mapping, given the types of entity attributes that are exchanged at model interfaces and the mapping specifications. Our methodology uses Event-B as the formal specification language and Discrete Event System Specification as the model composition framework; utilizes refinement relations between Event-B machines for the specification, verification and generation of the data conversion steps between models and employs a code generator that inputs the Event-B machine definitions to generate converter code that connects two model ports.

Keywords: multi-resolution modelling; model composability; DEVS; Event-B; modelling and simulation

Received 19 August 2013; revised 3 September 2014
Handling editor: Mariangiola Dezani-Ciancaglini

1. INTRODUCTION

Constructing complex software systems through the integration and coordinated interactions of simpler components has long been the focus of many research efforts. As the complexity of software systems increases, the need for systematic approaches increases accordingly. The issue at hand has a number of more specific problem areas, each of which deserves a distinct line of research. In the work presented in this paper our interest lies in a particular manifestation of this problem which is constructing complex simulation applications by composition. We focus on a relatively subtle but an important challenge for such complex simulation applications, which are cross-resolution simulations involving multi-resolution models [1]. Although different viewpoints may be attached to the associated set of problems by different research communities, we argue that the issue of resolution mapping pertaining to externally visible properties (i.e. in terms of inputs and outputs) of cross-resolution models is essentially a software engineering problem. In fact, the concept of *connectors* [2] in the context of Component-Based Development [3] and the concept of *adapters* [4] in the context of Service-Oriented Architecture [5] can be seen as more general cases of *resolution converters* as introduced in this paper. Resolution conversion has peculiarities that are mainly characterized by the multiple facets that induce distinct variations in the handling of the more general problem; these facets will be briefly discussed below. Not surprisingly, those peculiarities also allow us to contemplate the more focused and concrete solutions. As such, based on the analysis and selective targeting of those peculiarities, we will argue that the process of specifying, verifying and generating those converters can be supported by the use of a convenient formal language to develop a rigorous software engineering methodology. Along these lines, our primary motivation at the outset of our research was to develop a well-defined (i.e. based on formal specifications) and repeatable methodology with necessary tool support that diminishes the shortcomings of *ad hoc* coding practices mostly adopted for such conversion tasks. Our solution covers aspects ranging from model and mapping specification to mapping verification and application construction by component composition.

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

To facilitate a better understanding of the concepts of multi-resolution modelling (MRM) and the peculiarities mentioned above, we first provide an introductory summary, followed by our key contributions. The reader is referred to Section 2 for further information on the multi-resolution concepts.

1.1. Cross-resolution modelling in the context of complex simulation applications

A modelling enterprise involves capturing the characteristics and behaviour of systems via mathematical or logical abstractions. Intrinsically, the same real-world system can be represented by various abstractions. Depending on the requirements of different stakeholders, those abstractions may particularize specific aspects of the entities being represented or they may delineate varying levels of structural or behavioural information across the same set of aspects. The level of detail at which system components and their behaviours are represented is generally agreed to indicate the *resolution* of a model that represents that system.

The issue of multiple resolution models pertaining to the same real-world system manifests itself in the following three distinct forms: (i) When constructing a model of the system, the *modeller* may need to develop multiple representations of the same system, each with a different level of resolution. This is known as MRM. (ii) When developing simulation applications, the *simulationist* may need to compose or orchestrate models with different resolution levels, paying particular attention to the resolution conversion requirements. This is known as *Cross-Resolution Modelling* (*CRM*). (iii) When running a simulation scenario and analysing the results of a simulation run, the *analyst* may need to be aware of the existence of multiple resolution models and take the possible cross-resolution mapping issues into account when interpreting the results.

The characterization of these issues and the problems that may be attributed to them have already been explored in the relevant literature. For instance, Powell [6] provides a semi-formal description of the central concepts such as resolution, aggregation, disaggregation and consistency maintenance. Davis *et al.* [7–10] give an in-depth coverage of the key terminology and discuss a wide range of aspects offering considerable insight into the complexity of MRM. An extensively studied problem in the literature is the need for resolution mapping in a CRM and setting. The root of this problem is that resolution is a relative concept and exhibits variations across all the attributes of a model. Thus, the term 'level of resolution' is not congruously applicable to all the models involved in CRM so model developers need a methodology to compose models with different levels of resolution. Another root difficulty lies in the fact that the term resolution refers to a multifaceted concept (Fig. 1) [8] and it is a formidable task to find a solution that is applicable to all facets. Of the six facets given in Fig. 1, the process, spatial and temporal facets are related to the internal behavioural logic of a model and therefore it is less convenient to deal with them. This is primarily because the model internals can be arbitrarily complex and typically depend on many artefacts starting from the conceptual models and requirements ending with the implementation techniques across the development life cycle of the model. Composing cross-resolution models based on these facets will require coordination between the model designers and developers during the development cycle, which, in general, is not practical, since models might be developed at different sites and at varying times, using a wide range of technologies, standards and tools.

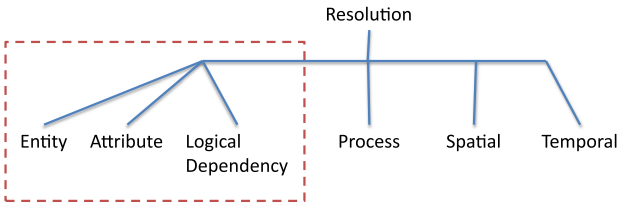

FIGURE 1. Aspects of model resolution (adapted from [8]).

The entity, attribute and logical dependency facets of resolution refer to the externally manifested properties of models such as the input/output variables and their data types. We claim that the resolution of a model can be more conveniently manifested by its visible properties. As such, we can document a model through its input/output variables and use this information to facilitate composition with other models of different resolution levels; however, to ensure a consistent data exchange between composed models, it is necessary to undertake a mapping of these variables. A purely syntactic approach to define the mappings between these variables may not result in a successful model composition, since the semantics of data types should be preserved through the resolution conversions. To cater for such semantics-preserving conversions, we propose to map the inputs and outputs of models via special connectors, called *converters*.

1.2. Our contribution

Clearly, *ad hoc* forms of converter building processes would not greatly contribute to the current state of CRM since many simulation application developers inevitably implement some form of conversion logic to compensate for the resolution mismatch between models. We argue that a rigorous resolution mapping methodology should be founded on a formal basis to address the four interrelated issues of (i) entity definition, (ii) mapping specification, (iii) verification and (iv) model composition. Furthermore, for practical purposes, this methodology should have tool support.

Our contributions to the work in the field of complex simulation software construction involving CRM are closely

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

aligned with the four aspects listed above. For item (i), we provide a methodology for resolution mapping in a CRM setting in which we use Event-B [11] as a formal specification language; we use *refinement relations* between Event-B machines for the validation and generation of the data conversion steps between models, to resolve (ii); we use tools to verify the transformations and provide a code generator that uses the Event-B definitions and refinement relations to generate the converter code, with the option of generating monitor code based on machine invariants in support of run-time verification to resolve (iii); we also propose a formal definition of a resolution converter in the Discrete Event System Specification (DEVS) setting, to incorporate it into a well-established model composition paradigm, and thus resolving (iv).

The remainder of this paper is organized as follows; Section 2 outlines the background to our research; Section 3 details our proposed approach, and Section 4 provides a description of the software tools to demonstrate the results of our work. Then, in Section 5, we present the discussions regarding the important aspects of this research. Finally, Section 6 contains our concluding remarks.

2. BACKGROUND

This section provides information that is essential for a better understanding of our work. First, we provide the definitions of multi-resolution and CRM, including the concept of aggregation/disaggregation, which is particularly important for entity resolution mapping. Then, we present introductory information concerning Event-B and refinement, which together constitute the formal basis of this research. Subsequently, we provide basic material on the DEVS formalism that constitutes the basis for our world view on model composability via port connections. Finally, we give a concise summary of SiMA, our simulation framework that implements an extended form of DEVS and serves as a basic software platform for the implementation of our CRM work.

2.1. Multi-resolution modelling

All simulation models are abstractions of a reality but some are more abstract, in the sense that they are less detailed than others that represent the same reality. Resolution is the level of detail at which system components and their behaviours are depicted [7]. The subject that deals with multiple levels of resolution for simulation models is called MRM.

A comprehensive definition of MRM is given by Davis [7] but it consists of the following basic points:

(i) building a single model with alternative user modes involving different levels of resolution for the same phenomena;
(ii) building an integrated family of two or more mutually consistent models of the same phenomena at different levels of resolution; or
(iii) both (i) and (ii).

For example, the sensor model in a wireless sensor network (WSN) simulation [12] can be implemented at different levels of resolution. To analyse a routing protocol, a sensor model with a simple battery and wireless model is sufficient; however, to analyse the monitoring capability of a sensor, details such as the sensing unit and its sensitivity to environmental conditions should be incorporated into the sensor model.

2.1.1. *Cross-resolution modelling*

CRM [13] is applicable to the concept of simulations at different levels of resolution that are required to interoperate. For CRM, it is important to understand the assumptions made concerning the levels of resolution of the simulation models. Two models that are required to work together might have different characteristics which would make interoperation difficult. Thus, at the heart of CRM is to ensure that such discrepancies are resolved, in order to allow simulations to interact with each other meaningfully.

To understand the CRM concept, a definition of resolution is required:

> As can be seen in Fig. 1, resolution is a multifaceted concept. Using a military example to make the distinctions, a higher entity resolution might mean following units as small as battalions rather than divisions; a higher attribute resolution might mean following the number of various weapons held by each battalion rather than merely assigning the battalion a net 'strength;' a higher logical-dependency resolution might mean including the constraints on the attributes and their interrelationships (e.g., the sum total of the men in the units comprising a division should equal the number of men in the division). A higher process resolution might mean computing the combat attrition at battalion level, rather than at division level and then spreading the attrition equally across the battalions in the division. A higher spatial and temporal resolution means using finer scales for space and time [8].

As seen in this definition of resolution, unless two models have been designed with CRM in mind, we cannot easily discuss the relative resolution between them, because it is likely that the models are complex and the resolution of one model compared with another can be higher in some respects, lower in others.

2.1.2. *Aggregation/disaggregation*

The common approach to CRM is aggregation/disaggregation and these twin processes ensure that entities interact with each other at the same level by forcing one entity to be formed at the level of the other. For example, in a WSN simulation [12] the sensors and interactions between sensors can be modelled in terms of single units or bundles. A bundle is an aggregation that models a set of sensors with a single base unit. If a low-resolution entity (LRE) and a high-resolution entity (HRE)

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

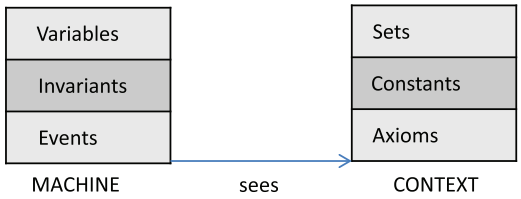

FIGURE 2. Machine and context (adapted from [14]).

need to interact, either the LRE will be decomposed into its constituents in a process known as disaggregation (LRE → HRE, bundles to units) or an aggregation process takes place (HRE → LRE, units to bundles).

2.2. Event-B

Event-B [11] is a formal modelling method for discrete systems based on refinement [14, 15]. The main purpose of creating models in Event-B is to consider and understand the complete system starting from an abstract description.

When modelling a system, Event-B creates the formal model, in such a way that the constant and variable parts are retained in the distinct components of *contexts* and *machines*, respectively. A machine consists of three distinct elements: (i) a set of state variables, (ii) a conjoined list of predicates, the invariants and (iii) some transitions, called events. A context consists of objects (sets and constants) and the axioms that constrain these objects (Fig. 2).

Events are operations that update the state variables of a machine. Each event is composed of guard and action statements. An event is allowed to execute an operation whenever all its guard statements return true. Action statements define the behaviour of the event operation and are required in order to update the state variables of the machine.

2.2.1. *Refinement*

Refinement [14] is applied to Event-B when there is an introduction of new machines and contexts that are related (refines, extends) to existing abstract ones (Fig. 3). The sets and constants of an abstract context are retained in its extension. In other words, the extension of a context only consists of new sets and constants. However, in the refinement of the machines, the concrete machine N has a collection of state variables that might be completely distinct from its abstraction M. However, it is allowed that invariants of N can depend on variables of its abstraction M, these are called *glue invariants* and they 'glue' the state of the concrete machine N to that of its abstraction M. We consider glue invariants to be important since we use them as the main constructs for the specification of the required data transformations between different entity resolution levels.

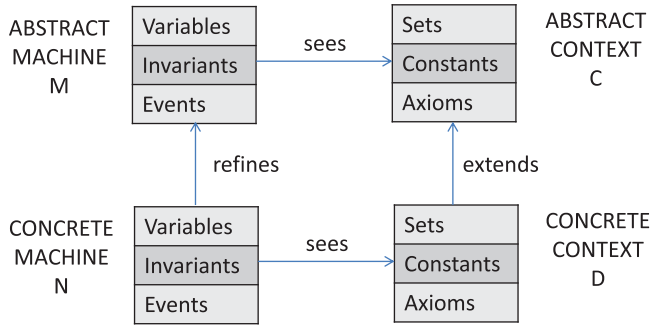

FIGURE 3. Machine refinement and context extension (adapted from [14]).

2.2.2. *Proof obligations*

To reason about a machine, we consider its proof obligations, which are produced from the union of invariants, axioms and guards [15]. For the purpose of our work, proof obligations serve to show that the glue invariants are consistent with the invariants of both machines.

There are several types of proof obligations, some of which are as follows:

(i) *feasibility*: The body of an event should not be blocked when the event is enabled, for example, the before/after predicates should not prevent from continuing;
(ii) *invariant preservation*: Action statements of events should preserve all the invariants of the machine.

A machine in Event-B must be verified by discharging its proof obligations. The following software tools enable both the definition and verification of Event-B machines:

(i) Atelier B [16] is a commercial product, designed primarily for B Language [17], but it has an extension for Event-B;
(ii) Rodin [18] is an open-source project which is actively being developed by its community. It has been developed over an Eclipse framework [19] and supports plug-in development for both functionality and language extension.

Owing to its power in application programming interface and since it has an open-source license, we selected Rodin as our Event-B tool.

2.3. Discrete event system specification

The DEVS is a formalism introduced by Zeigler [20] to describe discrete event systems. About 15 years later, a revision to DEVS was introduced called Parallel DEVS [21], which enabled the exploitation of parallelism in modern computers. In this formalism, there are two types of models; atomic and coupled. The former embodies behavioural logic and the latter is composed of other models, called sub-models,

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

and the connections between those sub-models. An atomic model in Parallel DEVS consists of a set of input events, state set and a set of output events; an internal, external and confluent transition function, an output function; and time advance function. The formal definition of an atomic model in the Parallel DEVS formalism is as follows:

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, ta \rangle,$$

where X is the set of input events; S is the set of states; Y is the set of output events; $\delta_{\text{int}} : S \to S$ is the internal transition function; $\delta_{\text{ext}} : Q \times X^b \to S$ is the external transition function, where X^b is the set of bags of the elements of X and:

$$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\} \text{ is total state set;}$$

$\delta_{\text{con}} : S \times X^b \to S$ is the confluent transition function, subject to $\delta_{\text{con}}(s, \phi) = \delta_{\text{int}}(s)$; $\lambda : S \to Y^b$ is the output function, where Y^b is the set of bags of the elements of Y; and $ta : S \to \Re_0^+ \cup \{\infty\}$ is the time advance function.

In DEVS, models communicate with each other using their ports, which are the interfaces of the models. External input events (X) are received by the input ports and the output events (Y) are sent from the output ports. The current state s is valid for a time interval, which is determined by the time advance (ta) function. After the completion of each time interval the output function (λ) is executed to send the output events that belong to the current state and then the internal transition function (δ_{int}) is executed to calculate the new state. If a model receives an external event during this time interval (e), the external transition function (δ_{ext}) is executed and the current state is updated to reflect the effects of the incoming events.

A coupled model is a composition of models (atomic or coupled models) and the port couplings between these models. Coupled models do not contain any behavioural logic, states or transition functions that need to be executed. They are intermediate structures that form the hierarchy in the model structure. A coupled model in the parallel DEVS formalism is formally defined as follows:

$$CM = \langle X, Y, D, \{M_i\}, \text{EIC}, \text{EOC}, \text{IC} \rangle,$$

where X is the set of input events; Y is the set of output events; D is the name set of sub-components; $\{M_i\}$ is the set of sub-components where, for each $i \in D$, M_i can be either an atomic DEVS model or a coupled DEVS model; EIC: external input coupling that connects external inputs to the sub-model inputs; EOC: external output coupling that connects sub-model outputs to the external outputs; and IC: internal coupling that connects sub-model outputs to the sub-model inputs.

A complete description of DEVS semantics can be found in [20–22].

2.4. Simulation modelling architecture

Simulation Modelling Architecture (SiMA) [23] is a modelling and simulation framework, based on the DEVS [20] approach to provide a solid formal basis for complex model construction. The SiMA simulation execution engine implements the parallel DEVS [21] protocol, which provides a well-defined mechanism for model execution. SiMA builds on a specialized and extended form of DEVS formalism, namely SiMA-DEVS, which:

(1) formalizes the notion of 'port types' leading to a strongly typed (and therefore type-safe) model composition environment. In this respect SiMA specializes the basic DEVS formalism by introducing type constraints on the port definitions;
(2) introduces the new *Direct Feed Through Transition* function, to account for model interactions that involve state inquiries with possible algebraic transformations (but with no state change) without a simulation time advance.

Strongly typed data ports require a model developer using SiMA to define data types to be used for inter-port communication. SiMA uses port data types for several issues including the serialization/deserialization of data values flowing between atomic models. For the implementation of our proposal, we employ SiMA as a modelling and simulation framework, and we use strongly typed ports to our advantage, but we do not use the direct feed through transition.

3. OUR APPROACH

The approach presented in this paper utilizes the data type information from the input/output variables used in models of different resolution levels to facilitate their composition. If two distinct models are to be composed via their input and output ports, either they must use identical input/output data types or a conversion between the data types is needed. Evidently, our assumption here is that composable models are *semantically related* but may have non-identical data representations due to the different entity resolution levels they employ.

To compose models at different resolution levels, our solution proposes the specifying, generation and use of converters between data types defining the ports of the connected models. Put simply, these converters are connectors between the output and input ports of models. Similar to models, converters have input and output variables, but unlike typical atomic models they do not have any behavioural logic. Contrary to the simplicity of the idea, achieving resolution mapping via converters following a systematic methodology is not straightforward. Thus, systematic and repeatable resolution mapping via converters needs to have the following three aspects:

(1) a formal language for specifying the entities to be mapped and the mapping between them;

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

(2) a tool that assists with verification of the conversion specification and generates executable converters for the conversion;
(3) a well-defined, formal basis for incorporating the notion of resolution converters into model composition schemes in a uniform way.

Although there are several proposals in [13, 24–26] regarding the use of converters between simulation models in the literature, to the best of our knowledge they fall short of offering a comprehensive solution with respect to the aspects listed above. The main novelties of our approach are as follows:

(1) It involves a formal proposal to fit the concept of converters into a well-established model composition paradigm (i.e. DEVS). Our proposal states that resolution conversion (of entities) can be specified uniformly via first class constructs called connectors that are inserted between couplings among atomic and/or coupled models in a DEVS setting.
(2) It provides a systematic methodology that offers a well-defined sequence of stages to obtain executable converters for entity resolution mapping, given the appropriate descriptions of entities and refinement relations. Our methodology relies on Event-B as a formal specification language that utilizes *refinement relations* between Event-B machines for the validation and generation of the data conversion steps between models, and employs a code generator that uses Event-B machine definitions and refinement relations to generate the converter code.
(3) It enables the systematic reuse of converters in a uniform way.

Section 3.1 provides a more detailed account of the first point given above; Section 3.2 gives details of the second point. Although the third point is a direct consequence of the first two, presenting a detailed discussion is outside the scope of the current paper.

3.1. The formal representation of converters in the DEVS setting

To forge the notion of converters into a coherent and well-established model composition paradigm, the formal representation of converters in DEVS terminology is important. The DEVS atomic model definition given in Section 2.3 provides a convenient formal basis for a converter definition.

Here, we propose that a *connector model* can be defined as a *standard DEVS atomic model* restricted in its time advance function, such that $ta : S \rightarrow \{0, \infty\}$ is the restricted time advance function.

The idea is that the atomic model operates in only two states; an idle state where the model waits for an input (i.e. the next time interval is equal to infinity), and a transition state with zero duration which is triggered by the receipt of an input that produces the converted output at the end of the state. Thus, the connector model does not violate the closure under the coupling property of DEVS, since it is essentially an atomic model.

Let cv denote an internal container for converted values. A converter can be implemented with the following transition functions:

(i) δ_{int}: An internal transition function that removes all converted values from cv and, due to implementation of ta, forces the model to wait for the next input, i.e. puts the atomic model into a passive or idle state.
(ii) δ_{ext}: An external transition function that converts each input value and stores the converted value in cv.
(iii) δ_{con}: A confluent transition function that calls δ_{int} first and δ_{ext} second as proposed in [21] as a default definition, i.e. $\delta_{\text{con}}(s, x) = \delta_{\text{ext}}((\delta_{\text{int}}(s), 0), x)$.
(iv) λ: An output function that gets the values from cv and publishes them to the output ports.
(v) ta: If cv contains converted values, the time advance function produces 0 to be able to dispatch output in the same simulation time; otherwise ∞ to let the model wait for an input.

3.2. A methodology for entity resolution mapping

Our methodology has five main stages:

(1) Definition of data types for the variables of Event-B machines (Section 3.3.1).
(2) Definition of mappings between data types of different resolution levels (Section 3.3.2).
(3) Verification of conversion steps (Section 3.3.3).
(4) Generation of the converter (Section 3.3.4).
(5) Optionally, instrumenting the converters with pre- and post-condition checkers as an aid for run-time verification (Section 3.3.5).

We present the details of our approach in the following subsections. Since our methodology supports both directions of mapping, first we will discuss HRE to LRE mapping. Subsequently, we will describe LRE to HRE and mixed mappings.

3.3. HRE to LRE mapping

To demonstrate our approach, we consider a simple WSN simulation [12] constructed using DEVS-compliant models (an extended version of our previous example as presented in [27]). A top-level visual representation of the DEVS models developed to implement the proposed WSN is given in Fig. 4. The WSN system consists of four components:

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

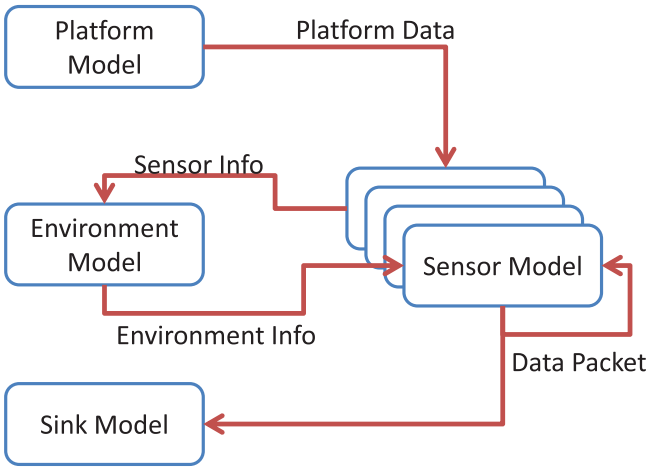

FIGURE 4. Models and entities in the sample scenario.

(i) *Sensors* detect the movement of objects in the environment and can communicate with other sensors within their range.
(ii) The *sink* unit is the base unit that collects and fuses all the information supplied by the sensor network.
(iii) The *platform* has a predefined path that it follows during the course of simulation. It represents the detectable object for the sensors in the environment.
(iv) The *environment* model calculates the environmental information depending on the sensor locations, such as the humidity and noise level that affect the signal transmission.

The simulation exercise is set to involve thousands of sensors spread over a wide area to track the path of a platform. This large number of sensors means that the simulation requires a large amount of CPU power; however, to decrease the CPU cost, the resolution level of some of the sensors that are away from roads can be reduced, since their behavioural requirements allow for lower fidelity levels. A low-resolution sensor model that was developed for previous simulations already exists, and is to be integrated into the current simulation.

The sensors transmit *DataPacket* values through their output ports. After introducing the low-resolution sensor models, we need to integrate a converter between the high-resolution and low-resolution sensors (Fig. 5).

The output data type the of the high-resolution model labelled *DetailedDataPacket* contains a 3D *Position* attribute with a *SignalStrength* value to be used by the receiver model for the receivable signal limit with environmental information from the environment model. The *Direction* value contains the vectoral position of the target. However, the output data type of the low-resolution model *DataPacket* contains a 2D *Location* attribute with an exact *CommunicationRange* value which is updated by a percentage with environmental information.

FIGURE 5. Composition of the new low-resolution model.

Furthermore, it contains only the *Distance* of the target without the direction.

3.3.1. *Using event-B for model data type definitions*

We use an Event-B machine as a container for the data type definitions for the input/output variables of a model. As mentioned above, an Event-B machine is specified using three main constructs, namely; a set of variables, a list of invariants and some events. We define a mapping of these constructs to three facets of the entity type specification in the following ways:

(i) Variables representing an attribute of an entity.
(ii) Invariants providing constraints on variables that are crucial for type conversions.
(iii) Events providing modifiers for variables; however, our methodology does not require any event definition.

The *DataPacket* entity can be represented with an Event-B machine as given in Fig. 6. Note that the VARIABLES section of the machine definition includes attributes of the complex type *DataPacket*; furthermore, the INVARIANTS section contains the types of attributes. In an Event-B machine, invariants have two major responsibilities:

(i) Data Type Specification: such as integer, string or complex types such as an array.
(ii) Provision of Constraints: The relations among variables are specified. For instance, range restrictions such as $x > y$ and $x < 1000$, or more complex constraints such as $x \leq y/z * 100$ can be stated.

Figure 7 shows the use of invariants for the provision of additional constraints for the *DataPacket* entity. These invariants are used for the dynamic checking of the converter output. As part of the code generation process Assertion statements are generated corresponding to those invariants.

3.3.2. *Applying Event-B refinement*

Traditionally, refinement is used to develop a concrete system based on an existing abstract model [14]. For the purpose of resolution mapping, a concrete system corresponds to a high-resolution (or disaggregated) entity, and an abstract system corresponds to a low-resolution (or aggregated) entity. Thus, refinement in this case can be viewed as obtaining a HRE from a LRE. The glue invariants of Event-B map the variables of a refined machine (i.e. the HRE definition) to those of an

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

MACHINE DataPacket
SEES DataPacketContext
VARIABLES

$CommunicationRange$
$Location$
$SensorLocation$
$PacketID$
$OriginID$
$TargetID$
$Type$
$Distance$

INVARIANTS

inv1 : $CommunicationRange \in \mathbb{N}$ (cm)
inv2 : $Location \in Vector2D$ (cm)
inv3 : $SensorLocation \in Vector2D$ (cm)
inv4 : $PacketID \in \mathbb{N}$
inv5 : $OriginID \in \mathbb{N}$
inv6 : $TargetID \in \mathbb{N}$
inv7 : $Type \in PacketTypes$
inv8 : $Distance \in \mathbb{N}$ (cm)
...

END

FIGURE 6. Event-B machine for the DataPacket entity.

inv9 : $Type = Activation \Rightarrow Distance = 0$
inv10 : $Type = Result \Rightarrow Distance > 0$
inv11 : $Distance < 2000$
inv12 : $CommunicationRange \leq 1000$
inv13 : $OriginID \neq TargetID$
...

FIGURE 7. More invariants for the Event-B machine of DataPacket entity.

abstract machine (i.e. LRE definition). This effectively makes the glue invariants the primary vehicle for the specification of the refinement relationships between two entities of different resolution levels. It should be the responsibility of the model developers to define the glue invariants for each refinement.

An example of the use of glue invariants for the purpose of defining a refinement from a *DataPacket* entity to a *DetailedDataPacket* entity (as depicted in Fig. 8) is illustrated in Fig. 9. Note that the *DetailedDataPacket* machine contains different variables and indicates, via the REFINES keyword, that it is a refinement for the *DataPacket* machine. It should also be noted that the glue invariants define how the aggregated variables of the *DataPacket* entity can be obtained from the variables of *DetailedDataPacket* entity. As a relatively subtle point, the use of accessor functions such as Vector3D_X for the variables of complex types can be seen in the expressions of glue invariants. These functions enable access to the members of complex data types (X member of a Vector3D variable) and are used for both the proof system and code generation described in the following sections.

3.3.3. Proving glue invariants

The verification of the glue invariants requires extensive tool support. Fortunately, for that purpose the Event-B community has developed such tools as Rodin [18], which generates proof obligations (POs) for possible gaps to be filled to construct the proof. Then, it attempts to automatically prove all POs and if this is not possible, it requests the user to prove the remaining POs manually. Inside the proof editor, Rodin requests the user to select related invariants about his/her model and produce the required statements to prove the proof obligation. If the current list of invariants is not sufficient to prove the PO, the user

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

MACHINE DetailedDataPacket
REFINES DataPacket
SEES DetailedDataPacketContext
VARIABLES

$SignalStrength$
$Position$
$SensorPosition$
$PacketID$
$OriginID$
$TargetID$
$Type$
$Direction$

INVARIANTS

inv1 : $SignalStrength \in \mathbb{N}$ (dB-microvolts)
inv2 : $Position \in Vector3D$ (cm)
inv3 : $SensorPosition \in Vector3D$ (cm)
inv4 : $PacketID \in \mathbb{N}$
inv5 : $OriginID \in \mathbb{N}$
inv6 : $TargetID \in \mathbb{N}$
inv7 : $Type \in PacketTypes$
inv8 : $Direction \in Vector3D$ (cm)
inv9 : $Type = Activation \Rightarrow ScalarLength3D(Direction) = 0$
inv10 : $Type = Result \Rightarrow ScalarLength3D(Direction) \geq 0$
inv11 : $SignalStrength \geq 40$
...

END

FIGURE 8. Event-B Machine for the DetailedDataPacket entity.

glue1 : $CommunicationRange = CalculateRange(SignalStrength)$
glue2 : $Location = mk\_Vector2D(Vector3D\_X(Position) \mapsto Vector3D\_Y(Position))$
glue3 : $SensorLocation = mk\_Vector2D(Vector3D\_X(SensorPosition) Vector3D\_Y(SensorPosition))$
glue4 : $Distance = ScalarLength3D(Direction)$

FIGURE 9. Glue invariants for the DetailedDataPacket entity.

should update his/her glue invariants or define more invariants to provide further constraints. Eventually, when all POs have been proved, the machine is verified and ready for converter generation.

For example, for the glue invariant *glue1* in Fig. 9 the prover requires the definition of *CalculateRange* function (e.g. SignalStrength * 10) in order to verify the $CommunicationRange \leq 1000$ and $SignalStrength \geq 40$ invariants. The related POs could be resolved by using the function definition inside the Rodin editor.

The Event-B prover is triggered by the statements in the event definitions of a machine. As our methodology does not require the modelling of events, the user needs to define an 'INITIALISATION' event with initial values of all variables. Although these values are not actual ones to be used in the simulation, they should be consistent with the constraints on both sides of the refinement.

For example, setting the initial value of the $SignalStrength$ variable to 40 will force us to set the $CommunicationRange$ variable to 400. If the $CalculateRange$ function were (e.g. $SignalStrength * 100$) we would make $CommunicationRange = 4000$, then the prover will warn us that the $CommunicationRange \leq 1000$ constraint is violated and we have to update our conversion routines.

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

3.3.4. *Converter generation*

Once the glue invariants that define the conversion relationships between the low-resolution and high-resolution entities are specified and proved to preserve the constraints (as specified by the glue invariants), then the converters can be generated based on the statements of the glue invariants. Note that converter generation does not depend on glue invariant proofs. Therefore, converters can be generated while the glue invariants are unproven, but such a practice would not be advisable from the viewpoint of reliability.

Glue invariants in general are of the form $x = \text{expr}(a, b, \ldots)$, where x represents a variable of a lower-resolution entity and $\text{expr}(a, b, \ldots)$ represents an algebraic expression in terms of variables, such as $a, b, \ldots$ of a higher-resolution entity. To generate the converter code, we transform these statements into statements in a programming language and inject them into the related methods that are called up during the simulation execution.

As such, the generation of a converter from *DetailedDataPacket* entity to *DataPacket* entity, for instance, fills the gap between our high-resolution and low-resolution sensor models, and allows the simulation to execute as expected (Fig. 5).

3.3.5. *Monitor generation*

As discussed in Section 3.3.1, the invariant statements about the provision of constraint information can be used to verify the input and output of the converters. They specify the relationships between variables and self-restrictions such as $x > y$ and $x < 1000$. We generate assert statements for each invariant that produces an error upon receipt of invalid values. At run-time the converter input/output is checked against the source and target machine invariants, respectively; more specifically, this occurs before and after the conversion takes place.

3.4. LRE to HRE mapping

Although, up to this point, we have discussed the mapping of HREs to LREs, our proposed approach puts no limit on the model composer with respect to the direction of the mapping. Event-B refinements can be put to work in both directions with refinement as usual and with abstraction.

The methodology stages, discussed thus far, can be applied to LRE to HRE mapping without any modification. Our transformation routines introduced above only employed aggregation, therefore additional information was not needed. However, in a CRM scenario, connections from LRE to HRE are likely to require additional information; thus resolving this issue requires further consideration.

To illustrate the problem in the context of our example described in Fig. 4, we require a converter from low-resolution sensor models to high-resolution models as the sensor network needs communication in both directions (Fig. 10).

FIGURE 10. Composition of a new low-resolution model with LRE to HRE mapping.

In general, the reverse direction requires extra information (to be input to the converter) or assumptions 'to fill the gap'. Assumptions to map the *CommunicationRange* value to the actual *SignalStrength* value can be produced by a model developer who knows the behaviour of the *high-resolution sensor model* and can interpret concrete values with respect to the abstract values.

For the example of the sensor model given above, we need to have a mapping from LREs to HREs. There are many ways to implement such mappings, but we can put them into two main categories:

(i) Mapping based on assumptions; a conversion function that computes the outputs for inputs based on the assumptions made by the model composer.
(ii) Mapping with the help of external data sources; the conversion function uses external data sources, such as statistical databases, to calculate the output.

These categories are described in the following sections.

3.4.1. *Mapping based on assumptions*

If the model composer knows the details of the high-resolution model behaviour and he/she can be certain of the required HRE values for the LRE values; thus, the model composer can implement a function such as $f(i_1, i_2, \ldots, i_m) = \{o_1, o_2, \ldots, o_n\}$ in which $i_1, i_2, \ldots, i_m$ are the low-resolution inputs and $o_1, o_2, \ldots, o_n$ are the high-resolution outputs.

For the *CommunicationRange* example it is possible to have a mapping similar to the one below:

(i) $CommunicationRange = 1 \rightarrow SignalStrength = 42$
(ii) $CommunicationRange = 2 \rightarrow SignalStrength = 44$
(iii) ...
(iv) $CommunicationRange = 1000 \rightarrow SignalStrength = 540$

3.4.2. *External data sources*

The *CommunicationRange* variable has a limited domain and can be implemented with simple mappings. More complex variables require external inputs such as statistical databases to fill the gap between the LRE and HRE values [7]. Hence, this reverse mapping of complex variables will require the model composer to attach external data sources to connectors to feed the transformation function.

Let E represent the external data source; then our mapping will look like $f(E, S, i_1, i_2, \ldots, i_m) = \{S', o_1, o_2, \ldots, o_n\}$.

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

axm10 : $CalculateRange \in \mathbb{N} \rightarrow \mathbb{N}$

axm11 : $\forall x \cdot x \in \mathbb{N} \land CalculateRange(x) \leq 10$

FIGURE 11. Sample function declaration and its output constraint.

The state variable S is a reflection of values previously converted and it will allow us to find more appropriate results from our data source.

For the sensor model example, the LRE side only transmits a *Distance* value, however, the HRE side requires a vectoral *Direction* value which requires the exact position of the target relative to the sensor at the time of detection. The LRE to HRE converter obtains the position of the target from a store that keeps the past values of all the outputs; then it fills the *Direction* value with the *Distance* computed from the *Position* taken from that store.

3.4.3. *Implementing mapping on Event-B*

Simple mapping functions such as those given in our examples can be easily implemented using Event-B machines. However, more complex functions that cannot be expressed using the language capabilities of Event-B require a different solution and, therefore, we suggest employing user-defined functions for that purpose. Event-B allows for the definition of a function declaration with its inputs and outputs and does not require other function details unless there is a proof obligation that needs to be proved. The constraints on inputs or outputs of such functions can be defined and implemented as shown in Fig. 11.

Although we cannot formally verify the implementation of these converter functions, we still have the option to validate their input and check their output at run-time with the generation of monitors as discussed in Section 3.3.5.

3.5. Mixed mapping

As discussed above, the resolution difference between two models might be mixed in the sense that some of the attributes within the same complex entity might be at low resolution and others at high resolution.

To illustrate this situation, we can add a new variable to our *WSN* example. The low-resolution sensor model uses the $TimeOfDay$ value of $DataPacket$ to calculate the actual $Distance$ because the sensing unit receives more noise during the day and $Distance$ should be reduced to find the target location. However, the developer of a high-resolution sensor model does not deal with the $TimeOfDay$ but uses $DayState$ which is an enumeration of *Night* and *Daytime*. Thus, our converter includes a mapping from $TimeOfDay$ to $DayState$ that involves aggregation into our LRE to HRE conversion.

As seen in our examples, both the HRE to LRE and LRE to HRE mappings can be specified for the generation of the same converter. Thus, our solution supports conversion in both directions for the same pair of models. If the model composer defines the required Event-B machine refinements, the converters will be generated accordingly.

3.6. Summary

The required stages to generate our converters are summarized as follows:

(1) Entity types are defined as Event-B machines.
(2) Constraints and conversion steps are defined using invariants.
(3) Machines (entities) that represent different levels of resolution are linked to each other with refinement relationships.
(4) All the machine invariants including glue invariants are proved.
(5) Programming language statements of the converter are automatically generated from glue invariants.
(6) Monitor statements that check the inputs and outputs of the converter are generated from invariants.

Section 4 contains an implementation of our approach using the SiMA [23] framework, followed by a discussion of our findings in Section 5.

4. IMPLEMENTATION IN A DEVS SETTING USING SIMA

As pointed out in Section 2.4, SiMA [23] is a modelling and simulation framework developed by our research group at TÜBİTAK BİLGEM İLTAREN. It is founded on DEVS [21] to provide a solid formal basis for building complex models through composition. SiMA provides a convenient software platform for our purposes since it inherently supports simulation construction through model composition. In that, the connectors fit well into the model coupling paradigm via the input–output ports. SiMA also comes with a tool-chain that facilitates the employment of a simulation construction methodology that involves a distinct stage in which all data types used for the input–output variables of model ports are defined in XML conforming to an XML Schema. In a later stage, automated code generation based on these types is achieved using the KODO tool. As explained in more detail in Section 4.2, this also is a good fit for our converter generation approach.

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

```
<event name="DataPacket">
  <member typeName="uint" name="CommunicationRange" type="primitive"/>
  <member typeName="Vector2D" name="Location" type="struct"/>
  <member typeName="Vector2D" name="SensorLocation" type="struct"/>
  <member typeName="uint" name="PacketID" type="primitive"/>
  <member typeName="uint" name="OriginID" type="primitive"/>
  <member typeName="uint" name="TargetID" type="primitive"/>
  <member typeName="PacketTypes" name="Type" type="enum"/>
  <member typeName="uint" name="Distance" type="primitive"/>
</event>
```

FIGURE 12. Sample KODO data type definition.

In addition to SiMA we also used Rodin [18] as an appropriate and easily accessible tool to create and verify Event-B constructs. Our development efforts contributing to our existing software infrastructure followed two paths:[1]

(1) Implementing additional tools for Rodin to cater for the definition of Event-B machines via KODO type definitions and the generation of source code from those definitions.
(2) Devising a mechanism for the incorporation of converters into the model composition, to enable the actual deployment and execution of those converters between SiMA models at simulation run-time.

For the first path, we developed two tools that operate as plug-ins to Rodin:

(i) *Event-B Machine Generator* (*EMG*): A tool to generate Event-B machines on Rodin that represent the KODO data types.
(ii) *Converter Code Generator* (*CCG*): A tool to generate converters from the refinement definitions of Rodin.

For the second path, we used the atomic model implementation provided by SiMA and treat atomic models as algebraic converters.

The overall converter generation process has four stages:

(1) Create KODO XML definitions for data types with different levels of resolution (i.e. including the abstract and refined counterparts of a certain entity).
(2) Use Event-B Machine Generator Plug-in of Rodin (EMG) to produce an Event-B machine definition for KODO data types.
(3) Decorate the Event-B machine definitions with glue invariants to specify the conversion expressions between abstract and refined entities.
(4) Use the code generator plug-in of Rodin (CCG) to produce source code that implements the conversion logic corresponding to those Event-B machine definitions.

These stages are described in more detail in the following sections.

4.1. Creating type definitions

As stated above, for a simulation construction we define our overall information space by creating XML definitions for entities flowing through the input–output ports of our models. This creates a natural precursor for potential Event-B machines where resolution conversion is needed. At this point we utilize the KODO tool within the tool-chain of SiMA to generate source code for input–output variable definitions for models. KODO has a well-defined schema and a type system to define data types in XML. KODO uses XML files that fully specify the data type definitions for the input–output variables to be used in SiMA models. These type definitions include primitive and complex types for each data field of objects as illustrated in Fig. 12.

4.2. Generating Event-B machines

In this stage, we use the EMG tool developed for Rodin that uses the KODO data type definitions to generate Event-B machine definitions. For example, the EMG reads the KODO type definition given in Fig. 12 to generate the Event-B machine in Fig. 13, which is, in fact, equivalent to the machine definition given in Fig. 6. Note that we generate some metadata in the form of comment tags (such as #Event, #Type, etc.) that allows us to determine the object types, invariant categories and other information during the code generation. The generated machine defines all fields of the KODO data type as variables, and their data types as primitive invariants. We also generate and use a *Context* called *MrmTypeContext* to define the type representations of all the complex data structures in Event-B.

[1]The full source code and work products, including case study implementation, can be downloaded from the first author's web page: http://www.ceng.metu.edu.tr/ e1565621/MRMCodeGen.

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

MACHINE DataPacket # Event
SEES MrmTypeContext
VARIABLES

$CommunicationRange$
$Location$
$OriginID$
...

INVARIANTS

inv1 : $CommunicationRange \in \mathbb{N}$ # TYPE
inv2 : $Location \in Vector2D$ # TYPE
inv3 : $OriginID \in \mathbb{N}$ # TYPE
...

END

FIGURE 13. Event-B Machine generated from the KODO data type definition.

```
protected override void ExternalTransitionWithObjects()
{
    foreach(var item in inputManager.GetIncomingEvents())
    {
        var newItem = outputManager.CreateNewEvent();
        ConvertItem(item, newItem);
    }

    this.HoldIn(0);
}
```

FIGURE 14. Sample external transition function.

4.3. Decorating the Event-B machines with glue invariants

The next step is the insertion of refinement relations in the form of glue invariants. The user is required to select the related data types to be used in the converters for model composition. Then he/she should define the *REFINES* keyword in the refinement machine (Fig. 8) and place his/her glue invariants as depicted in Fig. 9. Note that some variables exist in both the LRE and HRE model and do not require a glue invariant.

Additional invariants that define constraints and are required for monitor code generation should be inserted manually. An example of such additional invariants is shown in Fig. 7.

4.4. Generating the converter code

In this stage, the Event-B machines generated by the EMG and finalized by the manual decoration process as described in Section 4.2 are ready for the generation of converters. We used the glue invariants in refined machines as the basis of our converters and other invariants were used for verification.

As SiMA is implemented with C# language [28], our code generator generates converters in that language. The converter code can then be compiled together with the other source code which implements the overall simulation.

4.4.1. *Converter implementation using SiMA*

The central idea is that converters are placed along the coupling connections of the input/output ports between models. To achieve this in a systematic and uniform way, we utilize an existing construct of DEVS, namely the atomic model. In its original form, this model is used to implement the behavioural logic of a simulation model as a stateful component capable of operating in both discrete-event and discrete-time paradigms. Here, however, we use a specialized and somewhat reduced form.

To achieve this, we define a special atomic model, called the *Converter Model*, that implements the *External Transition Function* (δ_{ext}), which executes the appropriate converter function for the received value of the input port (see Fig. 14 for an example of the δ_{ext} code) and implements all other transition functions as described in Section 3.1.

4.4.2. *Converter generation*

After the generation of the model described in Section 4.4.1, the next stage is the generation of the converter code. Our *Converter Model* contains the *ConvertItem* function that lists the conversion statements for each field of the destination data type based on their glue invariants (Fig. 15).

A glue invariant has the form $x = f(y, z, \ldots)$, where x is the destination data type and $y, z, \ldots$ are sources. Details

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

```
private void ConvertItem(DetailedDataPacket source, DataPacket dest)
{
    MonitorSource(source);

    dest.Distance = ScalarLength3D(source.Direction);
    dest.Location = (new Vector2D((source.Position).X, (source.Position).Y));
    dest.OriginID = source.OriginID;
    ... // mappings for other variables of DataPacket event

    MonitorDestination(dest);
}
```

FIGURE 15. Sample glue converter function.

```
Debug.Assert(item.Type != PacketTypes.Result || item.Distance > 0,
                "DataPacket: inv8 verification error");
Debug.Assert(item.Distance < 2000, "DataPacket: inv9 verification error");
Debug.Assert(item.OriginID != item.TargetID,
                "DataPacket: inv11 verification error");
... // monitor functions for other DataPacket event variables
```

FIGURE 16. Sample content for a monitor function.

of f are defined as statements in the Event-B Machine or as function definitions in its Context. So, the code generation for the converter function depends on the specification of f and its implementation patterns. We adopt the approach of [29] regarding the target patterns in C# language and generate statements in the glue invariants (Fig. 9) as statements in C# language (Fig. 15).

4.4.3. *Compensating for the shortcomings of Event-B type system*

Event-B itself is not a fully fledged programming language, for example; it does not have primitive mathematical functions (e.g. sin, sqrt etc.); however, some glue invariants require these functions and to compensate for this shortcoming, our methodology supports the manual implementation of complex glue-functions.

To elaborate further, for a glue-statement such as $x = f(y) + q(z)$ we can generate full details of f and leave the implementation of q to the user. The C# language supports *partial class* definitions that allow the user to define his/her functions in different files. Our code generator generates f in the main file and requires the user to implement q in another file to be used in the converter.

Although this extension is instrumental in allowing the definition of refinements involving complex expressions, it limits our automatic verification capability since we can no longer use the implementation details of q in our proofs. However, since this extension is implemented due to the limitations of Event-B language, it can be omitted if, in the future, the Event-B language is enriched to support our requirements.

4.4.4. *Using invariants for run-time verification of converter output*

Allowing the manual insertion of a conversion code appears to be a loophole in our automatic verification capability. However, our Event-B machines involve invariants that provide constraints on the variables and the code generator can use these invariants for converter input and output checking.

We generate the assert statements as given in Fig. 16 and place them inside the *MonitorSource* and *MonitorDestination* methods and call them at the beginning and end of the *ConvertItem* method given in Fig. 15. These assert statements can be executed in the debugging phase of the software development and they can be omitted in the released executable.

5. DISCUSSIONS

In this section, we present a discussion of the central aspects of the work that is relevant to our solution.

5.1. Discussion of related work

Different approaches to the composition of multi-resolution models [24–26] have already been proposed in the literature. Among those, a relatively recent work, [25] introduced the concept of 'MR modelling space' to separate the aggregation/disaggregation (i.e. resolution conversion) logic from the mechanics of the simulation execution (i.e. the simulation space). Part of this Multi-Resolution Space is the Multi-resolution Event Interface (MREI), which handles the resolution mismatches of messages between models. In fact, the

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

idea of logically separating the resolution conversion and the simulation by localizing the entity resolution conversion into MREI has similarities with our approach in which the main difference lies in formalizing the notion of entity resolution conversion as a part of connector models in a DEVS setting. Furthermore, and more importantly, we address the reliability of the resolution conversion, firstly, through the formal verification of the Event-B glue invariants and, secondly, through the automatic generation of resolution converter components (i.e. connector models) from declarative specifications (i.e. Event-B machines). In that respect, it is important to note that we specifically target the problem of CRM as opposed to MRM. The implication of this emphasis is that our solution tackles the problem of that interoperation of models that are coupled via I/O ports exchanging objects at different resolution levels but representing the same real-world entities. Those models, possibly at different resolution levels, that can replace each other in a composition can be considered as members of a Multi-Resolution Model Family (MRMF). In fact, at a given simulation time only one member of this MRMF may be operational where it would have to interoperate with other models (that may possibly be a member of another MRMF); this in turn may require a resolution conversion process due to the difference in the entity resolution levels of the coupled ports. Postulating along the same lines, the problem of replacing a simulation model with another higher- or lower-resolution model that is a member of the same MRMF at simulation run-time (dynamically) can be addressed separately from the viewpoint of interoperation among cross-resolution models, although the ramifications of the two problems are related.

Apparently, the most relevant works to our approach are [13, 30] in which the authors present a number of what they refer to as 'fundamental observations' regarding the problem of CRM. We have found those observations quite useful in determining the qualities and level of comprehensiveness of the solutions in this field. We will not go into the details of each of their observations at length, rather we will compare central tenets of their solution with ours, namely, the Multi-Resolution Entity (MRE). Since the authors claim that MREs offer a solution framework that focuses on the maintenance of consistency based on their fundamental observations, we will undertake an informal evaluation based on a qualitative comparison of the MRE solution with our proposed solution as given in the following three subsections.

5.1.1. *Consistency maintenance*

MREs internalize the consistency maintenance via the management of a set of core attributes and a set of reversible mapping functions. MREs maintain 'internal consistency' across multiple, concurrent levels of resolution. Within the MRE concept, each entity either maintains state information at all desired levels of resolution or produces attribute values

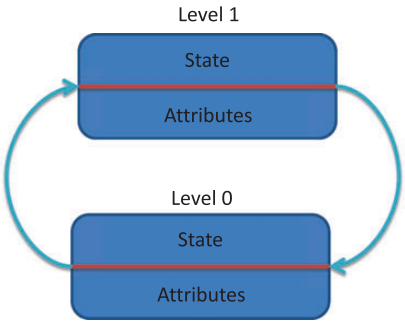

FIGURE 17. Design of an MRE with two resolution levels [13].

at each simulation step. Simulations involving MREs are based on concurrently reflecting the effects of interactions at all resolution levels. Figure 17 depicts a typical MRE for two levels; Level_0 for low resolution and Level_1 for high resolution. The MRE maintains the attributes at both levels at all times and the consistency between the two states of the MRE is maintained.

To maintain this consistency, relationships between the attributes must be captured. These relationships can be modelled by a directed, weighted graph where the nodes represent the attributes and the edges represent relationships. The MRE proposes the notion of an Attribute Dependency Graph (ADG) [30], which depicts the various attributes and sub-entities of the MRE and portrays the relationships among them. An ADG is an encoding of the concurrent multi-resolution interactions problem, and it is also an encoding of solutions thereof.

Our solution is similar to the MRE in many ways; however, it has the following additional advantages:

(i) Event-B machines that are linked to each other through refinement relationships can be considered to collectively define a Multi-Resolution Entity. From another perspective, the concept of a MRMF is embodied by the collection of Event-B machines that are linked via the refinement relationships.
(ii) The glue invariants that define the conversion logic between the attributes can be viewed as a specification of a directed graph between those attributes. In fact, the edges of such a graph are annotated with expressions that effectively specify a mapping between the related attributes.
(iii) If there are additional invariant definitions other than the glue invariants (such as range restrictions and type definitions) in the machines, those invariants are also taken into consideration by the prover during verification, which provides a further consistency checking mechanism on the resolution mapping.
(iv) One important advantage of our approach is that since our mapping is specified using a formal language, its

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

consistency can be verified using a prover; thus, its implementation can be generated automatically, which addresses the concept of correct implementation of the mapping.

(v) In our approach the consistency maintenance logic is internalized (and hence localized) into the notion of connector model; this ensures a systematic approach to the simulation construction and an effective management of consistency issues at run-time. Consistency maintenance in the sense of [30] amounts to the preservation of the glue invariants in our terms.

It is worth noting that our solution perceives simulation models as black-box components. The relation between the state and I/O of a model is crucial for the comparison between our solution and the MRE. If the data flowing through the I/O ports is a direct mapping of the internal model state, the implementation of converters would be equal to the state mappings of the MRE. Otherwise, we would lose some of the information about the state and our converter might not achieve the envisioned success rate of the MRE in consistency maintenance.

5.1.2. Staging and intrusiveness of resolution mapping procedures

Although the authors [30] do not extend their discussion towards design and implementation issues of MREs, our understanding is that the MRE is more intrusive in terms of the model internals since an MRE and the logic required to map the different resolution levels specified within it seem to be coupled with the behavioural logic of the implementation of the relevant model. This implies that a change in the structure (i.e. syntax) of an entity enforces complementary modifications on the internals of the models that consume or produce that MRE. This is something that may raise questions regarding the architectural clarity and maintainability aspects of the overall scheme when it comes to the actual construction of simulations based on this concept.

Our approach separates the definition of MRE families from the logic required to map between those MREs. This separation is clearly expressed at a declarative level using a formal language. Moreover, our methodology ensures that the data constraints and mapping logic are verified via provers, and implementation is generated, in an automated way, to support correctness by the construction. The expressive power of the language used (i.e. Event-B) does impose certain restrictions on the complexity of the logic that handles the resolution mapping; however, these restrictions are clearly delineated and there are ways of working around them.

5.1.3. Practicality

Since the authors [30] only provide a design strategy, rather than a complete solution with guidance for the implementation of the MREs, there are no details given concerning how consistency maintenance can be achieved in a simulation and execution of ADG.

Our solution provides a methodology involving both a design strategy and detailed implementation guidance. We give details of the stages required to apply our methodology and present an example of a complete converter generation. Hence, we describe a practical process to implement a CRM solution based on formal methods and available tools.

5.2. Converters in relation to connectors in component-based development

The use of connectors in component-based development is a well-known topic in software engineering [3, 31]. There is also a considerable amount of research on developing a formal basis for connectors in the context of component composition [2, 32–35].

The use of the Connector Model as a first class construct in a DEVS-based simulation construction environment is a specific case of the generalized connector concept as discussed in the literature cited above. In [36, 37], the authors propose drivers that catch incoming real-time events from hardware devices and send output commands to the hardware by user-implemented driver objects. These drivers can be viewed as connectors that allow the DEVS models to interoperate with real-time systems.

In our case, the connectors play the specific role of entity resolution conversion. Note that, although the adaptation of interaction protocols between components being 'glued' is one of the central properties of connectors in general, in our case this is less of an issue. This is because the interaction protocol of the DEVS models is under the strict control of the simulation engine that rigorously applies the DEVS simulation protocol [21]. Through this protocol the engine drives the models via a standard control interface that is, by definition, provided by each model. Therefore, port couplings between models only act as data flow channels. Since our connectors are defined to exist along port couplings (rather than between the engine and the models), their functionality is limited to data conversion.

It is worth noting that the use of Event-B to facilitate the verification and automatic construction of connectors in the DEVS setting to overcome resolution mismatches is a novel aspect of our methodology in terms of the application of formal methods in the component-based development of multi-resolution simulation software.

5.3. Converters in relation to adapters in service-oriented architecture

The notion of adapters is prevalent in the context of web service interoperability and adaptation. A service interface defines the set of operations that the service provides along with the associated message formats and data types. In our

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

case, we are only concerned with data types, modelled as Event-B machines. Let A and A′ be two 'functionally equivalent' services with different interfaces; then suppose that a service B is designed to use service A in a service composition. Now, if we need to replace A with A′ in the composition, a mismatch is likely to arise. In the SOA literature various categories of common mismatches are identified [4]. Overall, mismatches occur at two levels; protocol and interface, and it is acknowledged that these two levels cannot always be isolated. Since we work within the DEVS formalism, using the parallel DEVS protocol in particular, protocol mismatches are avoided. The interface level mismatches can be signature and parameter constraint mismatches. The signature mismatch is related to mismatches between operation names and parameters. In our case, the operation is implicit in the model ports selected in the model composition. In the case of a parameter mismatch, e.g. differences in the names, numbers and orderings of parameters, this is handled by the glue invariants. The parameter constraint mismatch is related to different constraints that the parameters must satisfy. In our work, this issue is addressed by the machine invariants. To sum up, adapters that handle interface mismatches between services are similar in purpose and functionality to the adapters proposed in this work.

Static schema matching approaches have been proposed that automatically match the interface specifications (in WSDL, in the form of XML documents) of web services [4, 38]. The techniques presented in the literature could be useful to complement the present work with the aim to automate glue invariant suggestion. The present work, however, focuses on the formal verification of glue invariants and generating adapter code automatically from them.

In [39], authors confront the interface mismatch problem by a sub-ontology extraction method. Their work focuses on the alignment of the extracted representation ontologies for the two service interfaces to be matched. In terms of our work, this corresponds to the entity resolution mapping stage. They mention adapter generation but no specifics are available.

In [40], the authors tackle the mismatch problem in heterogeneous plug-n-play devices using an ontology-based device and service description layer. They resolve the heterogeneity in the device ontology layer with semi-automatic alignment techniques. After validation by an expert they use the alignment and generate code for an executable proxy (converter). In terms of our work, alignments correspond to glue invariants and expert validation can be regarded as the informal counterpart of glue verification with POs. Although their overall approach is similar to ours, the semantics constraints, thus, the issue of preservation thereof, do not arise in their setting.

The use of formal methods in the SOA literature pertaining to the specification and generation of adapters has been limited. In [41], authors propose an application adaptation framework based on Petri nets used to capture template matchmaking using a reachability analysis. These templates are somewhat similar to our converters; they form the solution stages to solve the mismatches. Their work focuses on selecting templates for a mismatch condition, rather than formulating their definitions. In contrast, our work enables the definition, verification and generation of converters (templates) based on Event-B, a formal specification language.

Event-B has been used in the formalization of service composition. For example, in [42], authors map the web service composition process defined with BPEL on Event-B. They define a translation of BPEL entities (including data types defined in WSDL) to Event-B entities and present a proof- and refinement-based approach for the formal representation, verification and validation of Web Services composition. Our approach differs from this work with the code generation process that would help the automation of connector usage.

5.4. The unconventional use of refinement in our approach

In its conventional sense, refinement is a process to derive concrete models from existing abstract models. As such, it is used to develop more concrete models that are closer to the implementation. Many tools such as code generators developed by the Event-B community operate on the most refined models, because refinement reduces the level of abstraction and increases the level of detail, both of which are desirable improvements in a system development process. In the work presented in this paper, we take a different perspective, in that, we use refinement to define the relations between existing entities and use the defined relation itself as the source for the generation of a converter code. Evidently, our scheme does not exclude cases where entities and refinement relations that specify resolution mapping logic are defined together at the simulation design stage. However, our solution can operate in a setting where there are entities that represent data types at different resolution levels that have already been implemented and are ready to be used in a simulation. In such a setting, we add refinement relations between those existing entities to specify how to compensate for the resolution mismatches between them in a formal language. As described in Section 3, refinement allows us to define fine-grained mapping expressions among attribute pairs and allows the use of a proof subsystem to validate mapping specifications for the preservation of constraints and consistency. Thus, we exploit the power of refinement but employ it in an unconventional way. It is also worth noting, once again, that we use the proof capabilities of the Event-B tools [18] in refinement to validate our converters for constraint preservation.

Recently, Hallerstede and Hoang advanced the method of 'interface instantiation' as a special form of Event-B refinement with a view towards decomposition [43]. They define an interface as a collection of external variables

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

(i.e. those that are shared by the sub-models of a model) with the associated invariants. Our notion of converter, as a special kind of connector, can be seen as a counterpart of their notion of interface with a view towards composition. In their setting the instantiation of the interface facilitates decomposition, whereas in our setting generating code from the converter specification facilitates model composition.

6. CONCLUSION

In this paper, we have presented our approach to implement a solution for the composition of multi-resolution models. The construction of cross-resolution simulations is a multifaceted and complex enterprise. Any attempt to devise elegant solutions that target all of the facets of such a complex problem is deemed to face enormous difficulties and likely to fall short of delivering a comprehensive remedy that addresses all of those facets. From the very start of our work our objective has been to target a closely correlated subset of those facets (in this case the entity, attribute and logical dependency facets) and provide a relatively complete solution for that subset. In that respect, our approach combines the strength of formal approaches and languages, with tool and framework support into a systematic methodology, to deliver a focused and in-depth solution. Clearly, from the formal approaches we sought one that facilitates precise, machine processable semantics; and through the systematic methodology and accompanying tool support we looked for a repeatable process that builds upon rigorous foundations and relieves the simulationist from adopting *ad hoc* practices.

To reiterate, the merits of our approach are that it;

(i) involves a formal proposal to fit the concept of converters into a well-established model composition paradigm, namely, DEVS;
(ii) provides a practical methodology that offers a well-defined sequence of steps to obtain executable converters for entity resolution mapping, given the appropriate descriptions of entities and refinement relations.

Our approach potentially facilitates the automatic discovery and reuse of resolution converters (i.e. connector models as black-box components) that have been developed for mapping requirements that were already addressed in earlier simulation exercises. A library of such connector models can be built and made available for projects that involve CRM. Since our simulation environment supports strongly typed port definitions for both the models and connectors, the construction of a model composition graph that involves semantically compatible but syntactically incompatible model ports can be achieved through appropriate combinations of connectors. In fact, our research group has planned future work that can realize such graph-building processes in a semi-automated way. In [44], we have proposed our preliminary work about connector composition and reuse in DEVS.

The use of Event-B in the composition of simulation components can be examined from the perspective of the Levels of Conceptual Interoperability Model (LCIM) [45]. The use of Event-B machines and invariants, as detailed in the present work, addresses syntactic and semantic levels of interoperability, where the structure and meaning of exchanged data are shared between the interoperating systems. The pragmatic level of interoperability further requires that the systems have an agreement on the use of exchanged data. This level can be addressed by the use of the refinement relations involving events as well. Addressing dynamic interoperability will require the consideration of the internal states of the simulation components along with the operational context of the composite simulation. This level can be addressed by the modelling of the states and transitions of the components, and the global context in which the data exchange takes place. We regard supporting higher levels of LCIM in multi-resolution model composition by the use of Event-B and other formal methods as a promising direction for future research.

REFERENCES

[1] Committee on Modeling and Simulation for Defense Transformation, National Research Council (2006) *Defense Modeling, Simulation, and Analysis: Meeting the Challenge*. The National Academies Press.

[2] Meng, S. and Arbab, F. (2009) Connectors as designs. *Electron. Notes Theor. Comput. Sci.*, **255**, 119–135.

[3] Jifeng, H., Li, X. and Liu, Z. (2005) Component-Based Software Engineering. In Hung, D. and Wirsing, M. (eds), *Theoretical Aspects of Computing—ICTAC 2005*, Lecture Notes in Computer Science 3722, pp. 70–95. Springer, Berlin, Heidelberg.

[4] Kongdenfha, W., Motahari-Nezhad, H., Benatallah, B. and Saint-Paul, R. (2014) Web Service Adaptation: Mismatch Patterns and Semi-Automated Approach to Mismatch Identification and Adapter Development. In Bouguettaya, A., Sheng, Q.Z. and Daniel, F. (eds), *Web Services Foundations*, pp. 245–272. Springer, New York.

[5] Papazoglou, M. and van den Heuvel, W.-J. (2007) Service-oriented architectures: approaches, technologies and research issues. *VLDB J.*, **16**, 389–415.

[6] Powell, D.R. (1997) Control of Entity Interactions in a Hierarchical Variable Resolution Simulation. Technical Report. Los Alamos National Lab., NM, United States.

[7] Davis, P.K. and Bigelow, J.H. (1998) *Experiments in Multiresolution Modeling (MRM)*. RAND Corporation.

[8] Davis, P.K. and Hillestad, R. (1993) Families of Models that Cross Levels of Resolution: Issues for Design, Calibration and

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

Management. *Proc. 25th Conf. on Winter Simulation, WSC'93*, Los Angeles, California, USA, pp. 1003–1012. ACM.

[9] Davis, P. K. (2000) Dealing with complexity: exploratory analysis enabled by multiresolultion, multiperspective modeling. *In Proc. of the 32nd conf. on Winter simulation (WSC '00)*, Orlando, Florida, pp. 293–302. Society for Computer Simulation International.

[10] Davis, P.K. and Tolk, A. (2007) Observations on New Developments in Composability and Multi-Resolution Modeling. *Proc. 39th Conf. on Winter Simulation: 40 Years! The Best is Yet to Come, WSC'07*, Washington D.C., USA, pp. 859–870. IEEE Press.

[11] Abrial, J. (2010) *Modeling in Event-B: System and Software Engineering*. Cambridge University Press.

[12] Yick, J., Mukherjee, B. and Ghosal, D. (2008) Wireless sensor network survey. *Comput. Netw.*, **52**, 2292–2330.

[13] Reynolds Jr, P., Natrajan, A. and Srinivasan, S. (1997) Consistency maintenance in multiresolution simulation. *ACM Trans. Model. Comput. Simul.*, **7**, 392.

[14] Abrial, J.-R. and Hallerstede, S. (2007) Refinement, decomposition, and instantiation of discrete models: application to Event-B. *Fundam. Inform.*, **77**, 1–28.

[15] Hallerstede, S. (2008) On the Purpose of Event-B Proof Obligations. In Börger, E., Butler, M., Bowen, J. and Boca, P. (eds), *Abstract State Machines, B and Z*, Lecture Notes in Computer Science 5238, pp. 125–138. Springer, Berlin, Heidelberg.

[16] ClearSy System Engineering (2013) *Atelier B 4 - User Manual*.

[17] Abrial, J. (1996) *The B-Book: Assigning Programs to Meanings*. Cambridge University Press.

[18] Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T., Mehta, F. and Voisin, L. (2010) Rodin: an open toolset for modelling and reasoning in event-b. *Int. J. Softw. Tools Technol. Transf.*, **12**, 447–466.

[19] The Eclipse Foundation (2013). http://www.eclipse.org.

[20] Zeigler, B.P. (1976) *Theory of Modeling and Simulation*. John Wiley.

[21] Zeigler, B., Kim, T.G. and Praehofer, H. (2000) *Theory of Modeling and Simulation*. Academic Press.

[22] Barros, F.J., Zeigler, B.P. and Fishwick, P.A. (1998) Multimodels and Dynamic Structure Models: An Integration of DSDE/DEVS and OOPM. *Proc. 30th Conf. on Winter Simulation, WSC'98*, Washington, D.C., USA, pp. 413–420. IEEE Computer Society Press.

[23] Kara, A., Deniz, F., Bozağaç, D. and Alpdemir, M.N. (2009) Simulation Modeling Architecture (SiMA), a DEVS Based Modeling and Simulation Framework. *Proc. 2009 Summer Computer Simulation Conf., SCSC'09*, Istanbul, Turkey, pp. 315–321. Society for Modeling & Simulation International.

[24] Hong, S.-Y. and Kim, T. (2007) A Resolution Converter for Multi-Resolution Modeling/Simulation on HLA/RTI. In Koyamada, K., Tamura, S. and Ono, O. (eds), *Systems Modeling and Simulation*, pp. 289–293. Springer, Japan.

[25] Hong, S.-Y. and Kim, T.G. (2013) Specification of multi-resolution modeling space for multi-resolution system simulation. *Simulation*, **89**, 28–40.

[26] Baohong, L. (2007) A formal description specification for multi-resolution modeling based on DEVS formalism and its applications. *J. Def. Modeling Simul.*, **4**, 229.

[27] Deniz, F., Alpdemir, M.N., Kara, A. and Oğuztüzün, H. (2012) Supporting dynamic simulations with simulation modeling architecture (SiMA): a discrete event system specification-based modeling and simulation framework. *Simulation*, **88**, 707–730.

[28] Hejlsberg, A., Wiltamuth, S. and Golde, P. (2006) *The C# Programming Language*. Addison-Wesley Professional.

[29] Méry, D. and Singh, N.K. (2011) Automatic Code Generation from Event-B Models. *Proc. 2nd Symposium on Information and Communication Technology, SoICT'11*, Hanoi, Vietnam, pp. 179–188. ACM.

[30] Natrajan, A., Reynolds, P. and Srinivasan, S. (1997) MRE: A Flexible Approach to Multi-Resolution Modeling. *Proc. 11th Workshop on Parallel and Distributed Simulation, 1997*, Lockenhaus, pp. 156–163. IEEE.

[31] Bureš, T. (2006) Generating Connectors for Homogeneous and Heterogeneous Deployment. PhD Thesis, Faculty of Mathematics and Physics, Charles University.

[32] Hoare, C. and He, J. (1998) *Unifying Theories of Programming*. Prentice Hall.

[33] Davies, J., Faitelson, D. and Welch, J. (2008) Domain-specific semantics and data refinement of object models. *Electron. Notes Theor. Comput. Sci.*, **195**, 151–170.

[34] Chen, X., He, J., Liu, Z. and Zhan, N. (2007) A Model of Component-Based Programming. In Arbab, F. and Sirjani, M. (eds), *International Symposium on Fundamentals of Software Engineering*, Lecture Notes in Computer Science 4767, pp. 191–206. Springer, Berlin, Heidelberg.

[35] Allen, R. and Garlan, D. (1994) Formalizing Architectural Connection. *Proc. 16th Int. Conf. on Software Engineering, ICSE'94*, Sorrento, Italy, pp. 71–80. IEEE Computer Society Press.

[36] Cho, S.M. and Kim, T.G. (1998) Real-Time DEVS Simulation: Concurrent, Time-Selective Execution of Combined RT-DEVS Model and Interactive Environment. *Proc. 1998 Summer Simulation Conf.*, Reno, Nevada.

[37] Moallemi, M. and Wainer, G. (2010) Designing an Interface for Real-Time and Embedded DEVS. *Proc. 2010 Spring Simulation Multiconference, SpringSim'10*, Orlando, FL, USA, pp. 137: 1–137:8. Society for Computer Simulation International.

[38] Benatallah, B., Casati, F., Grigori, D., Nezhad, H. and Toumani, F. (2005) Developing Adapters for Web Services Integration. In Pastor, O. and Falcão e Cunha, J. (eds), *Advanced Information Systems Engineering*, Lecture Notes in Computer Science 3520, pp. 415–429. Springer, Berlin, Heidelberg.

[39] Jin, L., Wu, J., Yin, J., Li, Y. and Deng, S. (2010) Improve Service Interface Adaptation Using Sub-Ontology Extraction. *2010 IEEE International Conference on Services Computing (SCC)*, Miami, FL, July, pp. 170–177.

[40] El Kaed, C., Denneulin, Y. and Ottogalli, F.-G. (2011) Dynamic Service Adaptation for Plug and Play Device Interoperability. *Proc. 7th Int. Conf. on Network and Services Management, CNSM'11*, Paris, France, pp. 46–55. International Federation for Information Processing.

[41] Popescu, R., Staikopoulos, A., Brogi, A., Liu, P. and Clarke, S. (2012) A formalized, taxonomy-driven approach to cross-layer

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014

application adaptation. *ACM Trans. Auton. Adapt. Syst.*, **7**, 7: 1–7:30.

[42] Ait-Sadoune, I. and Ait-Ameur, Y. (2009) A Proof Based Approach for Modelling and Verifying Web Services Compositions. *2009 14th IEEE Int. Conf. on Engineering of Complex Computer Systems*, Potsdam, Germany, June, pp. 1–10.

[43] Hallerstede, S. and Hoang, T.S. (2014) Refinement of decomposed models by interface instantiation. *Sci. Comput. Program.*, **94**, 144–163.

[44] Kara, A., Oğuztüzün, H. and Alpdemir, M.N. (2014) Heterogeneous DEVS Simulations with Connectors and Reo Based Compositions (WIP). *Proc. 2014 Spring Simulation Multiconference*, SpringSim'14, Tampa, FL, USA, pp. 291–296. Society for Computer Simulation International.

[45] Tolk, A., Turnitsa, C. and Diallo, S. (2008) Implied ontological representation within the levels of conceptual interoperability model. *Intell. Decision Technol.*, **2**, 3–19.

Downloaded from http://comjnl.oxfordjournals.org/ at Carleton University on November 18, 2014