Using OCL for expressing temporal validity constraints*

Juliana Küster-Filipe and Stuart Anderson
School of Informatics
University of Edinburgh
The King's Buildings, Mayfield Road
Edinburgh EH9 3JZ
{jkfilipe|soa}@inf.ed.ac.uk

October 15, 2003

Abstract

In distributed real-time applications tasks on different nodes and components may require access to the same data. In the case of data replication, the data is duplicated on several components and procedures have to exist to make sure that the local copies of replicated data are kept *temporally consistent*. Further, different components in the system may have different *temporal validity* constraints for the same data, and as long as these constraints are satisfied overall system inconsistency is not harmful.

We propose the use of a formal analysis technique for guaranteeing temporal validity of replicated data. At the design level, we express general timing constraints as well as constraints on the temporal validity of data in UML's Object Constraint Language (OCL). To make this possible we provide a simple *timed-enriched liveness template* for OCL. Further we can translate these OCL constraints into logical formulae. The logic is a real-time temporal logic of knowledge suitable for verification through model checking. It allows us to check that the shared data in the system is consistent "enough" and cannot be a source of failure. We illustrate the approach with an open dynamic real-time distributed system.

1 Introduction

In distributed real-time applications where data needs to be shared among distributed components it is desirable to have overall data consistency at all times. It can be important in particular for safety-critical systems, where inconsistency can lead to catastrophic failures.

*Work reported here was supported by the EPSRC grant GR/R16891 and partially supported by the EPSRC grant GR/N13999.

Typical examples of safety-critical systems include systems in health-care environments, nuclear power plants, air traffic control and industrial automation. Both [11, 12] give several examples of software-related safety failures. In what follows, we are particularly interested in the analysis and design of real-time systems. In particular, these systems may require considerable human intervention.

For real-time systems it also makes sense to think about the *temporal validity* of shared data. Data can be considered to have a limited temporal validity. This because data may change over time and consequently its usefulness changes as well. The older the data gets the more unusable and unreliable it becomes. If the data is crucial for a human operator to perform a critical task, then to maintain the freshness of the data becomes vital. For example, consider systems which are continuously or periodically monitored by humans. If old data is displayed then the system is actively misleading the human (who reacts according to the monitored data) and contributing to a hazardous scenario. However, rather than continuously updating the data all we need is to guarantee data freshness in accordance to its temporal validity. In addition, it can be that different components in a distributed system need to have access to the same data but under different temporal validity constraints. This means that maybe one component in the system is always accessing older data than another component. Inconsistency in these cases will not be harmful as long as we can guarantee that the different temporal validity constraints are always locally satisfied.

Traditionally, the way to deal with this is to define scheduling mechanisms in such a way that higher priority is given to a component that needs fresher data [10, 5]. However, it may be the case that data requirements are not periodic or known in advance. This means that we would need *dynamic* scheduling mechanisms instead. These are not necessarily so easy to establish.

We are mostly interested in a more abstract view of the problem for analysis and design (thus laying the constraints that a scheduling mechanism at the implementation level has to satisfy). Notice that we are neither focusing on how to move from analysis and design to implementation, nor on efficient implementation mechanisms for maintaining temporal coherency of data directly. Other work, for instance [8], shows how to decompose system-level timing constraints (these include separation constraints which correspond to what we call temporal validity constraints here) into a schedulable set of fully periodic tasks. Also [18, 17, 20] explore for examples such as stock trading, how to implement efficient push-pull mechanisms guaranteeing dissemination of data in accordance to temporal validity constraints.

By contrast, there seems to be little work on explicitly considering temporal validity of data in analysis and design of real-time systems or even in more formal approaches. Furthermore, it is recognised that there is only a limited support for timing and schedulability analysis by current design tools and methodologies [19]. It strikes us, however, as a crucial aspect to guarantee the absence of some sources of failure. Moreover, the notion of temporal validity also loosens the assumption of overall data consistency in the system from *at all times* to within *local time frames*.

We propose the use of a formal analysis technique for determining the temporal validity

of shared data in real-time distributed applications. The approach consists of using an extension of UML's constraint language OCL at the design level to express needed temporal validity constraints. The required temporal extension to OCL is very simple and consists only of a *timed-enriched liveness template*. The template allows the designer to formulate required temporal validity (and other timing) constraints easily. Further, these OCL constraints can be translated into logical formulae. The logic used is a real-time extension of a temporal logic of knowledge.

Temporal logics of knowledge are logics which combine temporal operators (*always* in the future, *sometime* in the future, *until* and so on) with modal knowledge operators of traditional modal logics of knowledge (or epistemic logics). In general terms, the advantage of a knowledge operator is that it allows one to make precise the knowledge that an element in the system has (it can be a process, object, component or subsystem). Indeed, modal logics of knowledge (and temporal logics of knowledge) are commonly used in formal approaches of distributed computing. Moreover, model checking results have been established for some of these logics with interesting applications, for instance the checking of security protocols in [3]. However, temporal logics of knowledge generally do not consider real-time. Our logic combines the real-time characteristics of a well-known logic TCTL (*Timed* Computation Tree Logic) [2] with the advantages of an explicit knowledge operator. Because the logic is kept simple, automated verification through model checking is feasible. With respect to the concerns of this paper, we can with our approach check that the shared data in the system is consistent "enough" and cannot be a source of failure.

We illustrate the approach with a system for which QoS requirements like performance and reliability are of importance. Moreover, temporal validity issues have in this example an impact on the dependability of the system (more concretely on the mentioned QoS requirements). We consider the ParcelCall[1] case study: a *parcel localisation system* which is being developed as an European research and technology development project. ParcelCall constitutes an open dynamic real-time distributed computer-based system.

This paper is structured as follows. In Section 2, we discuss briefly temporal validity aspects in design. We then introduce a simplified description of ParcelCall. How temporal validity aspects for ParcelCall can be specified in OCL is explained in Section 4. Section 5 describes the formal knowledge-based framework. The logic is a real-time extension of a temporal logic of knowledge interpreted over timed automata. We finish the paper with some concluding remarks.

2 Temporal Validity in Design

Since the Unified Modeling Language (UML) was adopted as a standard modelling language by the OMG in 1997, the applicability of UML for real-time systems has been widely considered (see for instance [6] among others). Even though UML's core concepts are not thought for dealing with real-time aspects, UML has been designed to be extensible. This means that UML [15] offers extension mechanisms (through the concepts of stereotypes,

[1] Publications and project description can be found at `http://www.parcelcall.com`.

tagged-values and constraints) to be able to express aspects which are not covered by the core constructs. Since version 1.4, UML introduced the notion of UML *profiles* (sensible collections of extension mechanisms for a particular domain) and thereafter several requests of proposals for UML profiles have been issued by the OMG to address important areas. For example, a *profile for Schedulability, Performance and Time* [14] has recently been approved and work is under way for a UML profile for *Modelling Quality of Service (QoS) and Fault Tolerance Characteristics and Mechanisms* [13].

Though as simple as temporal validity may sound it does not seem to be addressed explicitly. In [6] Douglass stresses that old data can lead to supplying misleading information, which in turn may lead to hazardous conditions. Temporal validity as we understand it is, however, not considered, nor is how to deal with it in analysis and design of real-time distributed systems. It is not a difficult matter, and there are several indirect ways to cover temporal validity aspects of data. We could for instance use tagged values, constraints or even more informally notes to give some sort of indication on the temporal validity of data necessary at different locations or components in the system. Though ideally we should capture it in a way that would allow us to do adequate satisfiability checks.

Alternatively we could during design already want to consider a data replication strategy and model it explicitly in UML, for instance using activity diagrams. In this way we could possibly model the Just in Time Real-Time Replication (JITRTR) algorithm given in [16]. But this algorithm only works for static systems which is a stronger assumption than we want to make.

We propose the use of a formal analysis technique for determining the temporal validity of shared data in real-time distributed applications. The approach we advocate consists in expressing temporal validity constraints on data in design using UML's Object Constraint Language (OCL). OCL currently does not support real-time constraints directly, and there is recent work on temporal and real-time extensions of OCL (for instance [4] and [7] respectively). Neither does, however, address temporal validity constraints as needed though such an extension can be quite simple for [4].

OCL allows us to describe invariants on classes and types; pre and postconditions on operations and methods. Temporal validity constraints always have to hold and can thus be understood as special kinds of invariants. It is an invariant which states within what time data has to be refreshed (or equivalently, within what time a new update operation has to follow the occurrence of the immediately preceeding one). This is indicated in Figure 1, where $\mathsf{new}_j()$ corresponds to an update operation on the value of data j at component C

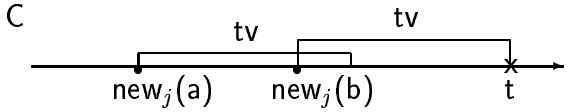

Figure 1: Temporal validity on data j.

with temporal validity tv (the new value is carried in the argument of the operation). In

order for the temporal validity constraint to hold, we know that the third $\mathsf{new}_j()$ has to occur before time t.

To be able to express data temporal validity constraints in some OCL extension has the advantage that we can easily translate the constraints into logical formulae to enable formal reasoning. In our case, it corresponds to a simple real-time logic of knowledge. Finally, this means that we can verify data temporal validity constraints (which are local to a component within a distributed system) with respect to the model of the system. We describe how approach works in Section 4.

3 The ParcelCall Example

The overall aim of the ParcelCall project consists of exploring the development of a low cost information infrastructure that enables the continuous information of the exact geographic position of parcels at any time. The result is thus a *parcel localisation system* which in the sequel we refer to simply as the *ParcelCall system*. The system corresponds to an open distributed system which is to be integrated with the legacy systems of transport and logistic companies. As a consequence of this integration, logistic or transportation companies (carriers) will be able to offer better and additional services to customers. For example, provide a customer the option to query the location and status of her transportation goods, more accurate delivery times, faster parcel delivery, and so on.

An initial ParcelCall system specification considered three main components:

- a *Mobile Logistic Server* (MLS): is an exchange point or a transport unit (container, trailer, freight wagon, etc). The transport units carry the parcels. Since containers can be inside other containers MLSs form a hierarchy. Main MLSs always know their current location via the GPS satellite positioning system.

- a *Goods Tracing Server* (GTS): comprises several databases which contains MLS hierarchies. Moreover, it keeps track of all the parcels registered in the ParcelCall system. GTS is also the component which is integrated with the legacy system of transport or logistic companies.

- the *Goods Information Server* (GIS): is the component which interacts with the customers and provides the authorised customer the current location of her parcels, keeps her informed in case of delivery delays, etc. Customers can interact with this component from a mobile phone (using WAP) or from a computer (using a web browser).

Figure 2 shows the architecture of ParcelCall in a UML-like notation: the three main components, some of their offered interfaces and component dependencies (others are omitted to simplify). For example, the GIS component offers two interfaces ILocalizeParcel and IParcelStatus. The first interface represents services offered to customers (e.g., a customer can start an interaction by querying the location of one of her parcels). The component CarrierSystem corresponds to the legacy system of a particular logistic or transportation

company. The CarrierSystem is a computer-based system and thus combines human carriers with several kinds of technological devices. According to the presented architecture, if a human carrier needs to reroute a parcel she will interact with GTS via the interface IParcelManagement.

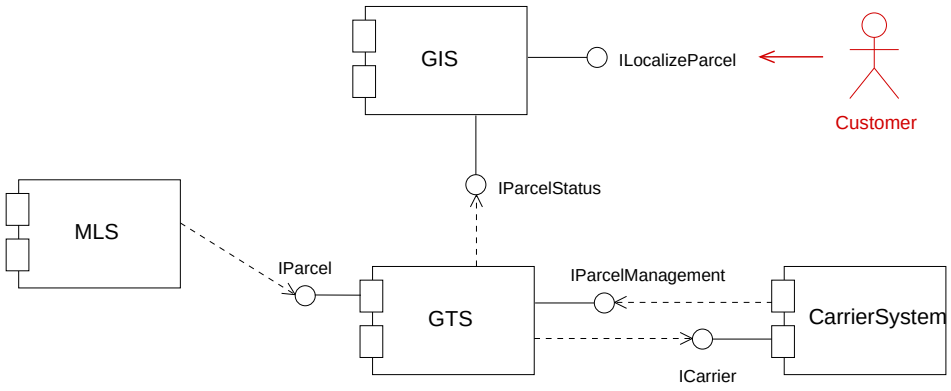

Figure 2: ParcelCall architecture.

The parcels itself are leafs in MLS hierarchies, and since containers and parcels can move between other containers these hierarchies are dynamic. Parcels have tags on them, and the ParcelCall project considered two possible kinds of tags: passive and active tags. We are herein interested in the active tags. Parcels with active tags know their current location via the GPS satellite positioning system, and send messages when their position changes.

The following are simplified assumptions we take here to illustrate temporal validity aspects:

- When the position of a parcel changes, the parcel (a MLS component object) sends a message (asynchronously) to the GTS component (via the interface IParcel).
- The GTS component knows for each registered parcel its delivery plan, its present location, its destination address, etc.
- The GIS component knows the registered parcels in the system, and the customers that are authorised to query their status and location.

Both GTS and GIS need to use the location data of parcels which must therefore be updated regularly. These components have different temporal validity assumptions on the data.

- GIS: The location of a parcel has a temporal validity which is not necessarily the same for all the other parcels. For instance, it may well be that a customer paid extra for being able to know the location of a parcel more accurately. In that case the data has a more limited temporal validity. We assume that there is a formula for calculating the temporal validity of a given parcel (more later).

- GTS: We assume that the temporal validity of location data for parcels is given by a constant t.

Notice that if the temporal validity constraints at GIS are always satisfied then we know that a customer will always have the right information on the status of his parcels and in accordance to his user tolerance. This denotes a QoS requirement on the system.

4 Expressing Temporal Validity using OCL

In [4] we defined an after-eventually (AE) template for describing certain temporal contracts.

context `Classifier`
after: `oclExpression`
eventually: `oclExpression`

Like an invariant or a pre/post-condition pair, an AE template is written in the context of a type, typically a classifier such as a class or a component from a UML model. As there, "self" may be used to refer to the instance of this type to which the contract is being applied.

The **after:** clause expresses some trigger for the contract: once the condition it expresses becomes true, the contract specifies a guarantee that the condition expressed in the **eventually:** clause will eventually become true. Notice that there is, in this particular example template, no intention that the **eventually:** clause should be required to become true immediately; the subtlety of this template is precisely that it is able to talk about consequences at a distance.

For expressing temporal validity constraints we want to restrict this template with respect to the time within which the **eventually:** clause is required to become true. We do so, by adding a third clause **within:** which expects a time value.

Coming back to the ParcelCall example. Assume the following

- In the MLS component, `Parcel` is a class with attributes `id` and `location`, and an operation `update()` which updates the value of `location`.
- In the GTS component, `Parcel` is a class with attributes `id` and `location`, and an operation `new(l)` which updates the value of `location` to `l`.
- In the GIS component, `Parcel` is a class with attributes `id`, `location` and `deliverymode` (of type natural number), and an operation `new(l)` which updates the value of `location` to `l`.

The following OCL constraint

context `MLS::Parcel`
after: `self.update()`
eventually: `GTS::IParcel.new(self.id,self.location)`

states that after a parcel (`self`) updated its location (after receiving a new signal from GPS) it will eventually notify GTS of the new location (meaning that the corresponding parcel at GTS will update its location value). Note that the :: notation is used in OCL to indicate a path, for example from a component MLS to a class Parcel. Also, `self` is essentially used as a variable and always refers to an instance of the contextual class (in this case Parcel at MLS). It can be omitted if the context is clear like in the examples below.

For the temporal validity constraints we have

context `GTS::Parcel`
after: `new(a)`
eventually: `new(b)`
within: `t`

for component GTS. It is essentially just capturing what was depicted in Figure 1. The constraint is very similar for GIS but instead of a time constant we need a formula for calculating the temporal validity of the data for a given parcel. As an example we considered the result of multiplying the value of `deliverymode` by a constant `o`.

context `GIS::Parcel`
after: `new(a)`
eventually: `new(b)`
within: `deliverymode` $\times$ `o`

5 Knowledge-based framework

The logics commonly used for reasoning about knowledge and time, in particular how knowledge can change throughout time, do not deal with real-time. By contrast real-time logics (for example TCTL [2]) do not capture local viewpoints and knowledge. We thus combine features of both into a simple linear real-time temporal logic of knowledge. The logic is interpreted over timed automata [1] (or more precisely unfoldings of timed automata). The semantics of the knowledge operator in this paper is the standard one for modal logics of knowledge (cf. [9]).

The syntax and semantics of our logic is described below. The syntax of the logic is illustrated by showing how the OCL constraints from the example can be written as logical formulae.

5.1 A real-time logic of knowledge

We consider that a distributed system consists of a finite set of components $S = \{C_1, \ldots, C_n\}$. Each of the components is understood as an independent process which can interact with other components in the system. The behaviour of a component is given by a timed automaton. The following definition is from [1].

Timed Automaton. A *timed automaton* A is a tuple $A = (S, S_0, \Sigma, X, I, E)$ where

- S is a finite set of *states*,
- $S_0 \subset S$ is a set of *initial states*,
- Σ is a finite set of *labels*,
- X is a finite set of *clocks*,
- I is a *mapping* that labels each state s with some *clock constraint* in $\Phi(X)$, and
- $E \subseteq S \times \Sigma \times 2^X \times \Phi(X) \times S$ is a set of *edges*. An edge $(s, a, \lambda, \varphi, s')$ represents a transition from state s to state s' labelled by a. φ is a clock constraint over X that specifies when the transition is enabled, and the set $\lambda \subseteq X$ gives the clocks to be reset with this transition.

For a set X of clocks, clock constraints $\varphi \in \Phi(X)$ are of the form

$$\varphi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi \wedge \varphi$$

where $x \in X$ and c is a constant in $\mathbb{Q}$.

A system model A is as usual obtained by parallel composition of models of the components $(A_1, \ldots A_n)$ according to their synchronisation laws. A *system run* r corresponds to a possible unfolding of the system model A and is herein considered to be given by

$$r \equiv (s_0, t_0) \xrightarrow{a_1} (s_1, t_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (s_n, t_n) \ldots$$

where $s_0, s_1, \ldots s_n$ are global system states (denoted by G) and $t_0, t_1, \ldots t_n$ are rational numbers denoting the time the system is in the corresponding state $s_0, s_1, \ldots s_n$. For every consecutive pair of states s_i and s_{i+1} somewhere in the system one or more components fired a transition a_{i+1}. The particular values of $t_0, t_1, \ldots t_n$ differ in different system runs but have always to conform to existing clock constraints (for instance t_i has to conform to $I(s_i)$).

We assume that there is a state projection function π that given a particular component j of the system ($j \in \{1, \ldots, n\}$) and a global state s returns the local state of the component in a system run r (denoted by $\pi_{r,j}(s)$).

Let w_r be a function that for a run r (as represented above) associates a global clock (an external clock which is not in the set of clocks of the system's timed automaton) to a global state of the system

$$w_r : \mathbb{Q} \rightarrow G$$

and is defined as follows

$$w_r(t) = s_i \quad \text{if} \quad \sum_{k=0}^{i-1} t_k \leq t < \sum_{k=0}^{i} t_k$$

A run only presents one possible computation of a system, and in general there are several possible runs for a system. Let $\mathcal{R}$ denote the set of all possible runs of a system.

We can now present the syntax of our simple logic.

Syntax of the Logic. Let $\mathcal{P}$ be a set of atomic propositions, and $\mathcal{A}$ be a set of action symbols. The formulae of our real-time temporal logic of knowledge are inductively defined as follows

$$\varphi := false \mid p \mid \varphi \Rightarrow \varphi \mid K_j\varphi \mid \langle a \rangle \varphi \mid \varphi \; \mathcal{U}_{\theta c} \; \varphi$$

where $p \in \mathcal{P}$, j denotes a system component from the set $\{1, \ldots, n\}$, a is an action symbol in $\mathcal{A}$, $c \in \mathbb{Q}$, and $\theta \in \{<, \leq, =, \geq, >\}$.

In particular for a distributed system component i, $K_i\varphi$ intuitively means that i knows φ from its local viewpoint (Note: the knowledge operator captures the notion of locality which is very important in a distributed system, where parts of the system may actually be unknown. It implies that we can do local reasoning instead of an impossible global reasoning).

$\mathcal{U}_{\theta c}$ is a bounded until operator where for example $\varphi_1 \mathcal{U}_{<c} \varphi_2$ informally means that φ_2 has to hold at a point in time less than c and until then φ_1 will always hold. The meaning of $\langle a \rangle$ is as usual in logics like the mu-calculus: the action a is the next transition to happen. Notice that we are assuming that actions have no duration, that is, actions are instantaneous.

An *interpretation structure* for a system is a pair $\mathcal{I} = (\mathcal{R}, \mu)$ where

$$\mu : \mathcal{R} \times \mathbb{Q} \rightarrow 2^{\mathcal{P}}$$

is a *valuation* function that associates to each moment in time the atomic propositions which are true at that moment in time.

Satisfiability for the logic is defined as follows

Satisfiability. Let $\mathcal{P}$ be a set of atomic propositions, and $\mathcal{A}$ be a set of action symbols. Let a system be represented by its set of runs $\mathcal{R}$ with $r \in \mathcal{R}$, an interpretation structure for the system be given by $\mathcal{I} = (\mathcal{R}, \mu)$, $x \in \mathbb{Q}$ and φ be a formula in our logic. The satisfaction relation $\mathcal{I}, r, x \models \varphi$ is defined inductively as follows

1. $\mathcal{I}, r, x \not\models \; false$
2. $\mathcal{I}, r, x \models p$ iff $p \in \mu(r, x)$
3. $\mathcal{I}, r, x \models \varphi_1 \Rightarrow \varphi_2$ iff $\mathcal{I}, r, x \models \varphi_1$ implies $\mathcal{I}, r, x \models \varphi_2$
4. $\mathcal{I}, r, x \models K_i\varphi$ iff for arbitrary $r' \in \mathcal{R}$ and clock y, if $\pi_{r,i}(w_r(x)) = \pi_{r',i}(w_{r'}(y))$ then $\mathcal{I}, r', y \models \varphi$
5. $\mathcal{I}, r, x \models \langle a \rangle \varphi$ iff $x = \sum_{k=0}^{j} t_k$ for some state (s_j, t_j) in r and $(s_j, t_j) \xrightarrow{a} (s_{j+1}, t_{j+1})$ and $\mathcal{I}, r, x + t \models \varphi$ holds with $x + t \leq \sum_{k=0}^{j+1} t_k$

6. $\mathcal{I}, r, x \models \varphi_1 \; \mathcal{U}_{\theta c} \; \varphi_2$ iff for some $t \; \theta \; c \quad \mathcal{I}, r, x + t \models \varphi_2$ holds and for all $0 \leq t' < t$ $\mathcal{I}, r, x + t' \models \varphi_1$ holds

The semantics is standard for most rules. Rule 5. essentially just says that $\langle a \rangle \varphi$ can only hold at a point in time when we are doing a transition labelled a and after a happens we reach a state where φ has to hold.

Some simple examples of things we can express in our logic are given in the context of the example introduced before.

5.2 The example revisited

We can express the previous OCL constraints by the following logical formulae (given in the respective order).

$$K_{MLS::Parcel}(\langle self.update() \rangle$$

$$true \; \mathcal{U}_{>0} \; (GTS :: IParcel :: new(self.id, q.location))$$

The above formula is stated from the local viewpoint of a parcel $self$ within MLS component. It reads literally as "after an update action occurred, then true holds until a message is sent to the IParcel interface of component GTS" (equivalent to sometime in the future of a standard non real-time temporal logic).

The temporal validity constraints make use of the bounded until, which contains in each case the temporal validity of the data in that component.

$$K_{GTS::Parcel}(\langle p.new(a) \rangle (true \; \mathcal{U}_{<t} \; \langle p.new(b) \rangle \; true)$$

$$K_{GIS::Parcel}(\langle p.new(a) \rangle (true \; \mathcal{U}_{<deliverymode \times o} \; \langle p.new(b) \rangle \; true)$$

In the first case, where the temporal validity is determined by a constant t, it says that after new occurs then in less than t units of time new has to occur again. Notice that here we have introduced a variable p to denote the contextual instance of class Parcel. The corresponding OCL constraints had left it implicit (we could also have used self or something else).

6 Discussion

In this paper, we proposed the use of a formal analysis technique for determining the temporal validity of shared data in real-time distributed applications. The approach suggests to use OCL in analysis and design directly (using a new template which allows one to express among others temporal validity constraints) and use a real-time temporal logic of knowledge for formal reasoning.

A natural advantage to use OCL (or some extension of it) to express temporal validity constraints on data is that the modeller does not need to know the details of the underlying logical formalism. Tool support can do the translation automatically. The logical formalism is useful because it enables formal reasoning. The presented logic can be used for automated verification through model checking (there is nothing in the logic which could present new difficulty in model checking as all operators are standard). We plan to build on top of existing model checking tools. Particularly suited is UPPAAL, a real-time model checking and simulation tool that has as underlying model timed automata and as logic a variant of TCTL.

Finally, the presented formalism allows us to check that the shared data in the system is consistent "enough" and cannot be a source of failure. In a broad sense a failure can mean that a particular quality of service requirement is not being met. This is particularly important if we are dealing with safety-critical systems where failures can lead to hazards. We therefore believe that our approach is a valuable addition to hazard analysis and hazard-control measures during early stages of software development of safety-critical systems.

References

[1] R. Alur. Timed automata. In *11th International Conference on Computer-Aided Verification*, volume 1633 of *LNCS*, pages 8–22. Springer, 1999.

[2] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[3] M. Benerecetti, L. Spalazzi, and S. Tacconi. Verification of the SSL/TLS protocol using a model checkable logic of belief and time. In S. Anderson, S. Bologna, and M. Felici, editors, *Proceedings of SAFECOMP'02*, volume 2434 of *LNCS*, pages 126–138. Springer, 2002.

[4] J. Bradfield, J. Küster Filipe, and P. Stevens. Enriching OCL using observational mu-calculus. In R.-D. Kutsche and H. Weber, editors, *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering (FASE), Grenoble, France, April 2002*, volume 2306 of *LNCS*, pages 203–217. Springer, 2002.

[5] A. Burns and A. Wellings. Real-time systems: Specification, verification and analysis. In *Advanced Fixed Priority Schedulling*, pages 32–65. Prentice-Hall, 1996.

[6] B.P. Douglass. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns.* Addison-Wesley, 1999.

[7] S. Flake and W. Mueller. An OCL extension for real-time constraints. In *Object Modeling with the OCL*, volume 2263 of *LNCS*. Springer, 2002.

[8] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, 21(7):579–592, July 1995.

[9] J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time, i: lower bounds. *Journal of Computer and Systems Science*, 38(1):195–237, 1989.

[10] M. H. Klein, T. Ralya, B. Pollack, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis*. Kluwer Academic Publishers, 1993.

[11] N. Leveson. *Safeware*. Addison-Wesley, Reading, Mass., 1995.

[12] P.G. Neumann. *Computer Related Risks*. Addison-Wesley, Reading, Mass., 1995.

[13] OMG. *UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms*, August 2002. Initial submission, available to members from www.omg.org.

[14] OMG. *UML Profile for Schedulability, Performance and Time*, May 2002. OMG document available from www.omg.org.

[15] OMG. *Unified Modeling Language Specification version 1.5*, March 2003. OMG document available from www.omg.org.

[16] P. Peddi and L.C. DiPippo. A replication strategy for distributed real-time object-oriented databases. In *Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'02)*. IEEE Computer Society, 2002.

[17] E. Pitoura, P.K. Chrysanthis, and K. Ramamritham. Characterizing the temporal and semantic coherency of broadcast-based data dissemination. In *International Conference on Database Theory*, 2003.

[18] K. Ramamritham. Temporally consistent delivery of time-sensitive information: Solutions and challenges. In *IFAC Conference on New Technologies for Computer Control*, 2001.

[19] M. Saksena. Real-time systems design: A temporal perspective. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, May 1998*, 1998.

[20] R. Srinivasan, C. Liang, and K. Ramamritham. Maintaining temporal coherency of virtual warehouses. In *19th IEEE Real-Time Systems Symposium (RTSS98), Madrid, Spain, December 2-4, 1998*, 1998.