Global Consistency Checking of Distributed Models with TReMer+

Mehrdad Sabetzadeh  Shiva Nejati  Steve Easterbrook  Marsha Chechik
Department of Computer Science
University of Toronto
{mehrdad,shiva,sme,chechik}@cs.toronto.edu

ABSTRACT

We present TReMer+, a tool for consistency checking of distributed models (i.e., models developed by distributed teams). TReMer+ works by first constructing a merged model before checking consistency. This enables a flexible way of verifying global consistency properties that is not possible with other existing tools.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications

General Terms
Design, Verification

Keywords
Distributed Development, Consistency Checking, Model Merging

1. INTRODUCTION

Models play a key role in many aspects of software engineering. Analysts build models of a problem domain to understand the stakeholders’ goals and needs, and they build models of a system under development to reason about its structure and behaviour. For large-scale projects, modelling is a collaborative effort that may involve distributed teams of people. These teams often build several inter-related models representing information from different perspectives, or information relevant to different development concerns. Model Management is concerned with describing the relationships between these models, and providing systematic ways to analyze and manipulate the models and their relationships [2].

An important activity in model management is checking the consistency of a set of models with respect to known or hypothesized relationships between them. For example, suppose that the class diagrams $M_1$ and $M_2$ in Figure 1 are two different perspectives on a GUI domain, and that the relationship $R$ in the figure expresses the overlaps between the two models.

In this example, $R$ was defined by hand. Larger problems require automation, usually achieved through conventions for establishing correspondence between object pairs, e.g., name equivalence if models have a common vocabulary, or identifier equivalence if models have common ancestors. For independently developed models, heuristic model matching has been proposed [6]. Regardless of how the relationship $R$ is defined, it is important to be able to check whether it respects the consistency properties of interest. For example, we may want to verify that $R$ does not imply cyclic inheritance. This is done using a rule like the following:

$$\neg \exists x, y, z, t \cdot R(x, y) \land R(z, t) \land \text{Descends}(x, z) \land \text{Descends}(t, y) \quad (C_1)$$

Obviously, the relationship in Figure 1 violates $C_1$, as witnessed by $x =$ "Component", $y =$ "UIObject", $z =$ "Canvas", and $t =$ "Panel". This indicates either that we misunderstood the nature of the overlaps between $M_1$ and $M_2$, or that there is a real disagreement between the models.

The example in Figure 1 only considers pairwise consistency, i.e., consistency of a pair of models with respect to a single relationship between them. In practice, we are often faced with systems that have many models inter-related by many relationships. Therefore, we not only need to check pairwise consistency, but also the consistency of a system as a whole. This is known as global consistency checking. Global consistency checking cannot be done through pairwise checking [14, 8]. To illustrate, consider the system in Figure 2.

The models in the figure are pairwise consistent when checked against $C_1$. But the system is globally inconsistent because the combination of the information provided by $R_1$, $R_2$, $R_3$ implies a loop in the inheritance chain. Specifically, Component inherits from Canvas $=_{R_1}$ Panel which inherits from Container (in $M_2$) $=_{R_2}$ Container (in $M_3$) which inherits from UIObject $=_{R_3}$ Component.

Existing tools for consistency checking of distributed models, e.g., xlinkit [7], focus on pairwise checking using rules similar to $C_1$. Unfortunately, such rules are coupled with explicit references to a relationship (e.g., $R$ in $C_1$). Hence, generalizing a pairwise rule to a global one involves referring to all of the relationships between...
the models within the scope of an individual rule, which makes the rules too complex to be practical.

In [11], we propose a technique for decoupling consistency rules from relationships when models are assumed to be homogeneous (i.e., specified in the same notation). The idea behind the technique is to employ model merging to reduce the problem of checking inter-model consistency to checking intra-model consistency of a merged model. For example, rather than trying to directly check the acyclic inheritance constraint on the system in Figure 2, we construct a merge, shown in Figure 6, and interpret the constraint over it. By keeping proper traceability data, we project the diagnostics obtained from consistency checking of this merge back to the source models and relationships.

In this paper, we describe a prototype tool, TReMer+, that supports the consistency checking approach in [11]. In addition to addressing the problem of global consistency checking for homogeneous models, TReMer+ offers practical insights on how to combine independent model management operators, in this case, merge and (intra-model) consistency checking, in order to perform more complex model management tasks.

2. TOOL OVERVIEW

Figure 3 shows an overview of the consistency checking use case in TReMer+. Having defined a system of inter-related models, we begin by applying a merge operator to the system. This yields a (potentially inconsistent) merged model along with traceability links from it to the source system. In the next step, we check the consistency of this merged model against the (intra-model) constraints of interest, and generate appropriate diagnostics for any violations found. By utilizing the traceability data for the merge, the tool enables navigation from the diagnostics to the source models and relationships involved in every inconsistency instance.

TReMer+ offers end-to-end support for the process in Figure 3. Specifically, it provides (1) an environment for building models, relationships, and systems of inter-related models, (2) a library of merge operators, (3) a platform for constraint checking and diagnostics generation, and (4) utilities for computation and navigation of traceability links. Of these, (3) and (4) are new in TReMer+; and (1) and (2) are from an earlier conception of the tool, TReMer (Tool for Relationship Driven Model Merging) [10].

In this paper, we describe the use of TReMer+ for consistency checking, concentrating on relevant parts of the model merging process (Section 2.2), specifying consistency rules (Section 2.3), and understanding the results of the analysis (Section 2.4).

2.1 Models, Relationships, and Systems

TReMer+ has a visual interface that allows users to edit their models, build relationships between these models, and define different systems by choosing subsets of the models and relationships in a project. TReMer+ currently supports entity-relationship diagrams, state machines, and simple UML domain models. In the future, we plan to extend TReMer+ to support other notations, such as goal models and detailed class diagrams.

TReMer+ provides a convenient way for describing relationships. For this purpose, model pairs are shown side-by-side. A correspondence is established by first clicking on an element of the model in the left pane and then on an element of the model in the right pane. Figure 4 illustrates this for the relationship $R_1$ of Figure 2.

2.2 Model Merging

Our consistency checking approach relies on the ability to merge a set of related models into a single model. The way this merge is carried out ultimately depends on the formalism being used, and the assumptions made about the semantics of the models and their relationships. To enable the implementation of different merge algorithms, TReMer+ defines a plugin interface for the merge operation. Currently, TReMer+ provides implementations for two merge algorithms, described in [9] and [6], respectively.

[9] is a generic algorithm suited to graph-based models in early stages of development (e.g., requirement elicitation), where models are typically incomplete and have informal or semi-formal semantics. The algorithm supports a range of ontological relationships between concepts in different models, e.g., exact matches (equivalence), specialization (isa), aggregation (hasA), and overriding (refutation of a concept in favour of another). Merges are computed based on a category-theoretic concept called colimit. Colimits can merge several models at a time; hence, they can be applied directly to interconnection diagrams with arbitrary numbers of models and relationships. Further, colimits are universal, i.e., unique (up to isomorphism) for a given interconnection diagram. Figure 6 shows
the colimit of the system in Figure 2. For simplicity, our example in Figure 2 only uses equivalence mappings. See [9] for more complex examples.

The second algorithm, [6], which is not illustrated here due to space restrictions, concentrates on state machine models with formal execution semantics. Unlike [9], it is assumed that models are complete. Relationships between model pairs are based on a notion of behavioural similarity between states. This notion is flexible enough to relate models that may be at different levels of abstraction. A merge is defined as a common refinement [13] of a pair of models with respect to a relationship between them. The approach generalizes to arbitrary interconnection diagrams by iteratively merging a new input model with the result of a previous merge. Since the merge operator in this approach is associative, the order in which binary merges are applied does not affect the final result. For examples and further details, consult [6].

2.3 (Intra-Model) Consistency Checking

TReMer+ uses CrocoPat [1] – a first order relational manipulation tool – for consistency checking of individual models. While our approach is not tied specifically to CrocoPat, there are two major considerations that make CrocoPat an appealing choice: (1) Expressiveness – CrocoPat provides the full expressive power of first order logic with counting and transitive closure. This offers great flexibility for specifying complex structural constraints. (2) Efficiency – CrocoPat uses symbolic encoding for relational structures, leading to high efficiency in terms of both time and memory.

Figure 7 depicts the process for consistency checking with CrocoPat. In the first step, the model to be checked is translated into a set of first order predicates using the translation algorithm given in [11]. The result, along with a user-selected set of consistency rules, is then passed to CrocoPat for consistency checking and generation of diagnostics. The diagnostics obtained, which are in hypertext format, are then sent to TReMer+ for presentation to the user.

Figure 6 shows the diagnostics generated when the merge in our running example is checked for cyclic inheritance. The diagnostics include a list of offending classes, and for each such class, a counterexample path. All model elements appearing in the diagnostics

In the above rule, we first compute a relation inherits(x,y) that holds if x inherits from y, i.e., if there exists an arc e of type ‘inheritance’ whose source is x and whose target is y (line 1). We then take the transitive closure (TC) of inherits(x,y). This yields a relation Descends(x,y) that holds if x is a descendant of y (line 2). Finally, we compute a set OnCyce(x) of classes lying on cyclic inheritance paths, i.e. classes that are descendants of themselves (line 3).

For diagnostics (lines 4–17), we iterate over the classes in OnCyce and print a cycle for each. Specifically, for every n ∈ OnCyce, we compute a set Admisible of n’s successors that have a path back to n (line 10). We choose an arbitrary element from Admisible (line

\[ \text{Invariant: } \bigwedge \text{Descends}(x,y) \implies \text{inherits}(x,y) \]
12) – in our code, the element with the smallest identifier (uid). After printing this successor (line 13), the process continues recursively, having removed from the inheritance relation the arc from \(n\) to the printed successor (line 14). The process ends when a full cycle is printed.

TReMer+ currently has rules for checking well-formedness of ER diagrams, UML domain models, and state machines. All these rules are specified in a single XML file which can be easily modified or extended by end-users. To simplify the specification of new consistency rules, TReMer+ provides a set of generic and reusable expressions capturing recurrent patterns in the structural constraints of graph-based models. These expressions are discussed in [11].

### 2.4 Inconsistency Navigation

To support projecting the inconsistencies found over a merge back to the source models and relationships, TReMer+ implements the traceability approach described in [9]. Specifically, for each element of the merge, traceability links are stored on the source elements and relationships relevant to that element. For example, the Projections pane in Figure 6 shows the links for Canvas, Panel in the merge. These include a link to Canova in \(M_1\), a link to Panel in \(M_2\), and further, a link to relationship \(R_1\) so that we know why Canova and Panel were unified.

In a typical inconsistency exploration scenario, the user starts from the diagnostics. Clicking a link in the diagnostics causes the corresponding element in the merged model to be highlighted and, at the same time, the traceability data for that element is displayed in the Projections pane. The traceability data, like the diagnostics, are in a hyperlinked format, allowing the user to navigate to the source models and relationships relevant to the element in question. Figure 6 shows the setting immediately after the user has clicked Merge/47 (i.e., Canvas,Panel) in the Diagnostics pane. If the user goes on to click R1/(uid:15, uid:19) in the Projections pane, she will be taken to the screen in Figure 4.

Our tool further provides traceability at the level of interconnection diagrams. For example, clicking on R1/(uid:15, uid:19) takes the user to the interconnection diagram in Figure 5(a) with R1 highlighted. This is useful when the user does not want to zoom into the details of the source models and relationships, and only wants to get a bird’s eye view of the models and relationships involved in a particular inconsistency instance.

### 3. EVALUATION

We have evaluated the practical utility of TReMer+ through two case studies. In the first study [11], we used a set of domain models for a health care system developed by the students of an advanced object-oriented modelling course. In the second study [6], we used variant specifications of telecom features from AT&T.

Both studies deal with independently developed models. For these models, relationship building is an exploratory process because one can never be entirely sure how the vocabularies of different models relate to one another. Each study begins by hypothesizing a set of relationships between the models, done manually in [11], and using a heuristic model matcher in [6]. TReMer+ is then employed for checking the structural consistency of these relationships, and the resulting diagnoses are used by the analysts to refine the relationships. Throughout the studies, we demonstrate that constructing merged models and checking global consistency enables various types of analysis that would be either expensive or impossible to do by pairwise consistency checking.

To ensure that global consistency checking scales, we have done a number of performance tests on models with 500 to 10,000 elements checking for such violations as dangling and parallel edges, multiple inheritance, and cyclic inheritance. The most computationally expensive of these checks ran in less than 3 minutes. Detailed results are available from [11]. From these results, we anticipate that our approach should be applicable to larger systems.

### 4. IMPLEMENTATION & AVAILABILITY

TReMer+ is written entirely in Java. It is roughly 15K lines of code, of which 8.5K implement the user interface, 5.5K implement the tool’s core functionality (model merging, traceability, and serialization), and 1K implement the glue code for interacting with CrocoPat. The tool uses JGraph [5] for editing and visualizing models, and EPS Graphics2D [4] for exporting models to PostScript vector graphics. TReMer+ was publicly released in May 2007. The tool and our case studies are freely available at [http://www.cs.toronto.edu/~mehrdad/tremer/](http://www.cs.toronto.edu/~mehrdad/tremer/).

### 5. CONCLUSION

We presented a tool, TReMer+, for detecting global inconsistencies in distributed models. The tool enables checking inter-model properties of a set of models via checking intra-model properties of their merge. TReMer+ is currently limited to homogeneous model only. Generalization to heterogeneous models presents a challenge because merge cannot be defined at a notational level. In future work, we plan to develop ways to merge models at a logical level and provide support for heterogeneous consistency checking [7, 3]. Another natural follow-on to our current work is resolution of global inconsistencies.

TReMer+ is part of a larger research effort to build usable model manipulation tools. A complementary part is a project aimed at developing a customizable Eclipse-based platform for model management. An outline of this project is given in [12].

### Acknowledgments

We thank everyone who provided feedback on TReMer+, particularly Dirk Beyer, Renée Miller, Rick Salay, and Pete McCormick. Funding for this work was provided by OGS, NSERC, and Bell Canada (through the Bell University Labs).

### 6. REFERENCES


APPENDIX

A. PLAN

There will be four parts to our presentation: (1) motivation and problem statement, (2) overview of solution and tool architecture, (3) case study, and (4) wrap-up. To present these parts, we use a combination of slides, flash demos, and a live demo. In particular:

- Parts (1) and (4) are presented using only slides.
- Part (2) is presented using slides and short flash demos that illustrate the tool over toy examples. We choose flash demos over a live demo for this part because they more easily weave into the flow of the slides as we explain the underlying methodology of the tool. To see examples of the flash demos we plan to use, visit the tool’s website.
- Part (3) is presented as a live demo based on the case study described in [11]. The demo is preceded by slides that motivate the case study and accompanied by slides (on the second projector) that highlight the main observations. The focus of the demo is to let the audience see the interactions with the tool first-hand, and to explain what TReMer+ offers over existing consistency checking tools.

Below, we provide further details about our presentation plan.

A.1 Motivation & Problem Statement

Estimated Duration: 5–6 minutes
Delivery: 4 to 5 slides

Motivation:
- With the increasing globalization of software development, management of distributed models has become a key problem.
- Central to this problem is the issue of checking consistency between different models.

Existing Work:
- Existing techniques focus on checking consistency of individual models and of pairs of models.
- Such strategy cannot verify global consistency.

Example of Global Inconsistency:
- This is going to be similar to the running example of the paper.

A.2 Overview of Solution & Tool Architecture

Estimated Duration: 6–8 minutes
Delivery: 2 to 3 slides, 2 short flash demos

Overview:
- We explain the consistency checking process in TReMer+ using the diagram in Figure 3. In this context, we also explain the components of the tool and how they fit together.

TReMer+ in Action:
We present two flash demos.
- The first one provides an end-to-end tour of the tool over a simple conceptual modelling example, e.g., the running example of the paper. The demo starts by defining models, relationships, and systems. It then goes on to compute a merge (using the merge operator of [9]), check the consistency of this merge, and project the resulting diagnostics.
- Our second demo illustrates consistency checking of distributed behavioural models described as Statecharts. The demo emphasizes TReMer+’s ability to work with different merge operators. The models we use in this demo are small fragments of the AT&T telecom specifications; and the merge algorithm we apply is the one described in [6].

A.3 Case Study

Estimated Duration: 9–11 minutes
Delivery: 2–3 slides, a live demo

Case Study – Setup and Goals
- Models: Five requirements domain models for a health care system
  - Models were developed independently by five individual students.
  - Models are small (with 50-70 elements each) but real. See Figure 8.
- Goals:
  1. Constructing a coherent set of relationships to express the overlaps between the models.
  2. Exploring the structural conflicts between the models.

Case Study – Execution
1. Hypothesize a set of preliminary relationships between the models.
2. Check the consistency of these relationships using TReMer+.
3. Use the resulting diagnostics to address the goals of the case study.

Live Demo – Scenario
In our demo, we concentrate on the second and third steps of the execution of the case study. Specifically:
1. We apply merge to an already-prepared interconnection diagram like the one shown in Figure 9. This results in a merged model like that in Figure 10.
2. We check this merged model against several structural constraints to obtain inconsistency diagnostics like those in Figure 11.
3. We select two or three inconsistencies from the diagnostics and explain how each of them can be used to reason about the source system (e.g., Figure 12). As we progress through the demo, we also explain our observations about the advantages of global consistency. A summary of these observations is provided below.

Live Demo – Observations
Collapsing several pairwise checks into one global check
- We can consolidate several checks that have a pairwise nature into a single global check. For example, suppose we have identically-named concepts in different models but the concepts are not equated by the relationships defined between the models. This is a potential anomaly and should raise a warning. One
could do this using pairwise comparison of the models, but a more effective way would be to construct a merge and report all identically-named elements in one shot.

**Detection of design conflicts**
- In our case study, there are concepts such as StaffMember and Schedule that appear in several of the source models; however, different subjects described the associations between these concepts differently. For example, one said Schedule belongs to StaffMember, another said StaffMember has a Schedule, a third one said StaffMember aggregates Schedule, and so on. Global consistency checking allows us to analyze all these conflicts within a unified context, i.e., the merged model. This is much more effective for understanding the conflicts compared to pairwise analysis of the source models.

**Decision impact analysis**
- We can use global consistency checking to analyze the impact of decisions made over one model on other related models. For example, if we decide to delete a concept \( C \) from some model, we can automatically check all other models which have a concept \( C' \) equated with \( C \), to verify that no (non-deleted) arcs are incident to \( C' \) – such links would become dangling if we propagated the deletion of \( C \) to \( C' \). Employing pairwise checking for performing such sanity checks after a change can be costly or even ineffective when more complex sanity criteria are involved.

An important issue to convey during the demo is that although merges are often large, they can still be successfully used for exploring inconsistencies. This is because, for every inconsistency, the user needs to focus only on the (typically small) merge fragment that is relevant to that particular inconsistency. Nevertheless, we recognize that exploration can be facilitated by showing to the user only this relevant fragment rather than the entire merge. This can be achieved by using a model slicing operator [2]. We plan to incorporate such an operator into our tool in the future.

### A.4 Wrap-up

*Estimated Duration: 1 minute*

*Delivery: 1 slide*

We conclude with a summary and provide download information for the tool.

### B. SCREENSHOTS

In the next four pages, we provide representative screenshots for our live demo.
Figure 8: Source models for the demo.

Figure 9: An interconnection diagram for relating the models in Figure 8. In addition to M1–M5, a helper model, Helper1, appears in the diagram. This model is used for bridging the information gaps between the original five models. Note that we did not have to state all relationships between model pairs, because merge automatically results in transitive relationships. For example, having a relationship that equates $x$ in M1 and $y$ in M2, and another relationship that equates $x$ in M1 and $z$ in M3, automatically equates $y$ and $z$. 
Figure 10: Merge of the system in Figure 9.
Figure 11: A fragment of the diagnostics generated for the merged model of Figure 10. The dialog box for consistency rule selection is also shown.
Figure 12: Example of reasoning about the source system – the marked diagnostic message is a warning on the existence of parallel association arcs between the concepts Ward and Nurse. From the projections, we determine the origins of these arcs: one comes from M2 and the other from M4. If the original arcs turn out to be expressing the same association, we need to declare a correspondence between them.